ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Desarrollo de herramientas didácticas para el aprendizaje de Symbian OS

AUTOR: Carlos José Piñera Sánchez
DIRECTOR: Javier Vales Alonso

Octubre / 2007

Autor	Carlos José Piñera Sánchez
E-mail del Autor	carlospsanchez@gmail.com
Director(es)	Javier Vales Alonso
E-mail del Director	jvales@upct.es
Codirector(es)	-
Título del PFC	Desarrollo de herramientas didácticas para el aprendizaje de Symbian OS
Descriptores	Symbian, Sony Ericsson P800, J2ME, C++, Carbide C++, Netbeans

Resumen

Debido al gran interés que suscitan en la sociedad de hoy en dia los dispositivos móviles, estos, han sufrido una gran evolución en los últimos años. Se han producido una gran cantidad de cambios, tanto en el hardware como en el software ampliando sus capacidades hasta unos límites inimaginables hace algunos años.

Este desarrollo ha producido la necesidad de nuevos entornos para los mismos, produciendo el desarrollo de nuevos sistemas operativos encargados de gestionar los dispositivos. Entre todos estos sistemas operativos destaca Symbian OS, ya que nació de un consorcio entre las grandes marcas de desarrollo de tecnología móvil (Nokia, Sony Ericsson, LG, Samsung, etc.) y por tanto ha sido apoyado e incluido en todos sus terminales mas avanzados.

Symbian OS es hoy en dia el sistema operativo para sistemas móviles mas expandido. Posee una gran cantidad de peculiaridades que lo hace atractivo para la gestión y desarrollo sobre smartphones. Por todo ello surge la necesidad de aprender a utilizar este sistema operativo.

En este proyecto se han desarrollado un completo tutorial de aprendizaje de Symbian/OS, que reúne y explica todas las herramientas que proporciona. Se han explicado pormenorizadamente cada uno de los componentes, con el fin de servir como base para el aprendizaje escalonada en el aprendizaje de las tecnologías involucradas. El conjunto de prácticas propuestas se desarrolla mediante herramientas de emulación, y a través de pruebas directas en el Smartphone de Sony Ericsson P800. Finalmente, se ha completado el proyecto con una discusión de las ventajas y posibles inconvenientes sobre cada uno de los componentes explicados.

Titulación	Ingeniería Técnica de Telecomunicaciones, Especialidad Telemática
Intensificación	-
Departamento	Tecnologías de la información y las comunicaciones
Fecha de Presentación	Octubre - 2007

Índice General

Índice de figuras

1 Introducción

2 Symbian OS

3 Java

4 C++

Capitulo 1

Introducción

1.1 Planteamiento

Actualmente el sistema operativo Symbian esta adquiriendo una gran importancia en el mercado de la telefonía móvil, proveyendo a los terminales de unas posibilidades y funcionalidades hasta ahora desconocidas. Esto esta haciendo que todas las grandes marcas estén involucrándose en el desarrollo e integración de este sistema en un gran número de sus productos.

Esta tecnología es relativamente joven y por ello es difícil encontrar información para poder comenzar a trabajar con ella. Además gran parte de esta información no tiene un formato claro y esta en ingles o en otras lenguas distintas del castellano, lo que complica aun mas la tarea de empezar a manejar un sistema diferente como el que se trata aquí.

Por ello, se decidió elaborar un dossier aunando toda la información encontrada, explicando las diferentes herramientas y lenguajes que existen para desarrollar software en Symbian OS. Intentando hacerlo con la mayor claridad posible, con una gran variedad de ejemplos de los dos posibles lenguajes que se puede utilizar en este sistema operativo (Java y C++) y con todas las herramientas existentes para el desarrollo de este software.

Se dividirá el mismo en dos bloques bien diferenciados. El primero de ellos intentará dar una explicación de las herramientas, librerías, clases, etc. en definitiva todas las armas que provee Java para crear software en Symbian OS. Se abordarán temas como los puntos fuertes y los puntos débiles de este lenguaje en dicho sistema e irán acompañados de unos ejemplos para una comprensión más sencilla. En el segundo bloque se realizará un desarrollo similar al primero, pero para un lenguaje de programación diferente C++.

Se pretende que esto sea una guía de iniciación y reconocimiento de este sistema, y que sirva para facilitar la tarea de todos aquellos que estén interesados en aprender a utilizarlo.

1.2 Sistemas Operativos

Un sistema operativo (SO) es un conjunto de programas destinados a permitir la comunicación del usuario con una computadora y gestionar sus recursos de manera eficiente. Comienza a trabajar cuando se enciende el ordenador, y gestiona el hardware de la máquina desde los niveles más básicos.

Un sistema operativo se puede encontrar normalmente en la mayoría de los aparatos electrónicos que podamos utilizar sin necesidad de estar conectados a un ordenador y que utilicen microprocesadores para funcionar, ya que gracias a estos podemos entender la máquina y que ésta cumpla con sus funciones (teléfonos móviles, reproductores de DVD... y computadoras).

1.2.1 Funciones básicas

Los sistemas operativos, en su condición de capa software que posibilita y simplifica el manejo de la computadora, desempeñan una serie de funciones básicas esenciales para la gestión del equipo. Entre las más destacables, cada una ejercida por un componente interno (módulo en núcleos monolíticos y servidor en microkernels), podemos reseñar las siguientes:

- Proporcionar comodidad en el uso de un computador.
- Gestionar de manera eficiente los recursos del equipo, ejecutando servicios para los procesos (programas)
- Brindar una interfaz al usuario, ejecutando instrucciones.
- Permitir que los cambios debidos al desarrollo del propio SO se puedan realizar sin interferir con los servicios que ya se prestaban.

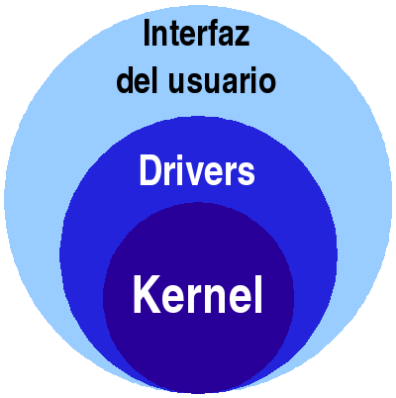


Figura 1. Niveles de un Sistema Operativo.

1.2.2 Perspectiva histórica

Al comienzo de la era informática, los sistemas no utilizaban sistemas operativos. Estas computadoras de hace 40 años ejecutaban un programa a la vez que era cargado por un programador. Este cargaba el programa y lo ejecutaba. Si existía algún error que hiciera que el programa se detuviera antes de lo esperado, se tenía que comenzar de nuevo con todo el proceso. Recordemos que en esa época no había muchas computadoras en funcionamiento, así que el programador tenía que esperar de varios días hasta tener nuevamente su turno enfrente de la computadora. Los sistemas operativos existen porque son una solución razonable al problema de crear un sistema informático útil. El objetivo fundamental de los sistemas informáticos es ejecutar los programas de los usuarios y facilitar la resolución de sus problemas. Todo esto se hacía a través de tarjetas perforadas que una persona encargada cargaba en la computadora y luego de algunas horas devolvía la salida impresa al programador.

Al avanzar la tecnología informática, muchos de estos programas se cargaban en una sola cinta, otro programa residente en la memoria de la computadora, cargaba y manipulaba los programas de esa cinta. Este es el ancestro de los sistemas operativos de hoy en día. En la década del 60 la tecnología de sistemas operativos avanzo mucho y se podían tener múltiples programas al mismo tiempo en la memoria. Así surgió el concepto de multiprogramación. Si un programa necesitaba esperar a que ocurriera algún evento

externo, como que una cinta se rebobinara, otro podría tener acceso a la CPU para así poder utilizar el 100% del poder de procesamiento con que contaba la computadora. Esto ahorraba mucho dinero ya que en aquel entonces todo en lo referente a cómputo (memoria, espacio en disco, etc) costaba cientos de miles de dólares. A finales de los 60's, en 1969, nació UNIX, SO que trataremos más adelante, y es la base de muchos de los sistemas operativos de hoy en día, aunque muchos no lo admitan.

1.2.2.1 Problemas de explotación y soluciones iníciales

El problema principal de los primeros sistemas era la baja utilización de los mismos, la primera solución fue poner un operador profesional que manejaba el sistema, con lo que:

- Se eliminaron las hojas de reserva.
- Se ahorró tiempo.
- Se aumento la velocidad

Para ello, los trabajos se agrupaban de forma manual en lotes mediante lo que se conoce como procesamiento por lotes (batch) sin automatizar.

1.2.2.2 Monitores residentes

Según fue avanzando la complejidad de los programas, fue necesario implementar soluciones que automatizarán la organización de tareas sin necesidad de un operador. Debido a ello se crearon los monitores residentes: programas que residían en memoria y que gestionaban la ejecución de una cola de trabajos.

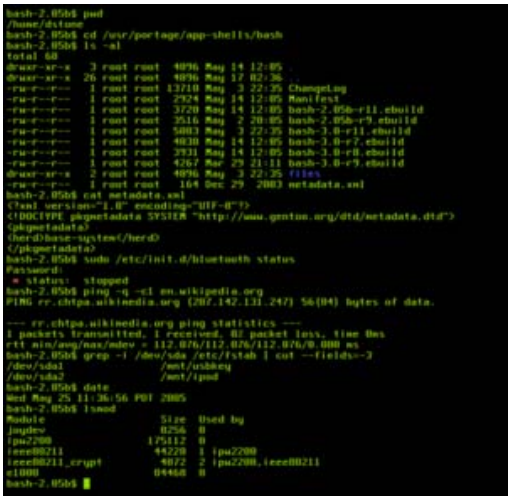


Figura 2. Consola (shell) de una computadora.

Un monitor residente estaba compuesto por:

- Cargador
- Intérprete de comandos
- Controladores (drivers) para el manejo de entrada/salida.

1.2.2.3 Sistemas con almacenamiento temporal de E/S

Se avanza en el hardware, creando el soporte de interrupciones. Luego se lleva a cabo un intento de solución más avanzado: solapar la E/S de un trabajo con sus propios cálculos. Por ello se crea el sistema de buffers con el siguiente funcionamiento:

- Un programa escribe su salida en un área de memoria (buffer 1).
- El monitor residente inicia la salida desde el buffer y el programa de aplicación calcula depositando la salida en el buffer 2.
- La salida desde el buffer 1 termina y el nuevo cálculo también.
- Se inicia la salida desde el buffer 2 y otro nuevo cálculo dirige su salida al buffer 1.
- El proceso se puede repetir de nuevo.

Los problemas surgen si hay muchas más operaciones de cálculo que de E/S (limitado por la CPU) o si por el contrario hay muchas más operaciones de E/S que cálculo (limitado por la E/S).

1.2.2.4 Spoolers

Hace aparición el disco magnético con lo que surgen nuevas soluciones a los problemas de rendimiento:

- Se eliminan las cintas magnéticas para el volcado previo de los datos de dispositivos lentos y se sustituyen por discos (un disco puede simular varias cintas).
- Debido al solapamiento del cálculo de un trabajo con la E/S de otro trabajo se crean tablas en el disco para diferentes tareas, lo que se conoce como Spool (Simultaneous Peripherial Operation On-Line).

1.2.2.5 Sistemas Operativos Multiprogramados

Surge un nuevo avance en el hardware: el hardware con protección de memoria. Lo que ofrece nuevas soluciones a los problemas de rendimiento:

- Se solapa el cálculo de unos trabajos con la entrada/salida de otros trabajos.
- Se pueden mantener en memoria varios programas.
- Se asigna el uso de la CPU a los diferentes programas en memoria.

Debido a los cambios anteriores, se producen cambios en el monitor residente, con lo que éste debe abordar nuevas tareas, naciendo lo que se denomina como Sistemas Operativos multiprogramados, los cuales cumplen con las siguientes funciones:

- Administrar la memoria.
- Gestionar el uso de la CPU (planificación).
- Administrar el uso de los dispositivos de E/S.

Cuando desempeña esas tareas, el monitor residente se transforma en un sistema operativo multiprogramado.

1.2.3 Llamadas al Sistema Operativo

Definición breve: llamadas que ejecutan los programas de aplicación para pedir algún servicio al SO.

Cada SO implementa un conjunto propio de llamadas al sistema. Ese conjunto de llamadas es el interfaz del SO frente a las aplicaciones. Constituyen el lenguaje que deben usar las aplicaciones para comunicarse con el SO. Por ello si cambiamos de SO, y abrimos un programa diseñado para trabajar sobre el anterior, en general el programa no funcionará, a no ser que el nuevo SO tenga el mismo interfaz. Para ello:

- Las llamadas correspondientes deben tener el mismo formato.
- Cada llamada al nuevo SO tiene que dar los mismos resultados que la correspondiente del anterior.

1.2.3.1 Modos de ejecución en un CPU

Las aplicaciones no deben poder usar todas las instrucciones de la CPU. No obstante el SO, tiene que poder utilizar todo el juego de instrucciones del CPU. Por ello, una CPU debe tener (al menos) dos modos de operación diferentes:

- Modo usuario: el CPU podrá ejecutar sólo las instrucciones del juego restringido de las aplicaciones.
- Modo supervisor: la CPU debe poder ejecutar el juego completo de instrucciones.

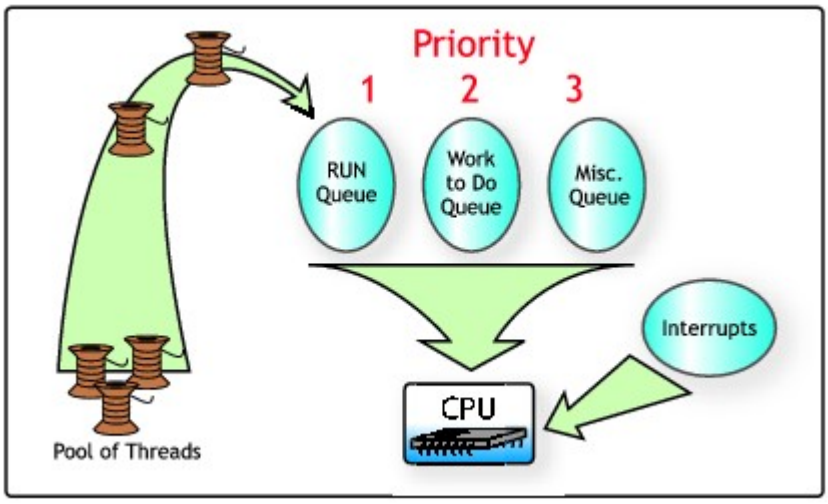


Figura 3. Prioridad de ejecución en una CPU

1.2.3.2 Llamadas al Sistema

Una aplicación, normalmente no sabe dónde está situada la rutina de servicio de la llamada. Por lo que si ésta se codifica como una llamada de función, cualquier cambio en el SO haría que hubiera que reconstruir la aplicación.

Pero lo más importante es que una llamada de función no cambia el modo de ejecución de la CPU. Con lo que hay que conseguir llamar a la rutina de servicio, sin tener que conocer su ubicación, y hacer que se fuerce un cambio de modo de operación de la CPU en la llamada (y la recuperación del modo anterior en el retorno).

Esto se hace utilizando instrucciones máquina diseñadas específicamente para este cometido, distintas de las que se usan para las llamadas de función.

1.2.3.3 Bibliotecas de interfaz de llamadas al sistema

Las llamadas al sistema no siempre tienen una expresión sencilla en los lenguajes de alto nivel, por ello se crean las bibliotecas de interfaz, que son bibliotecas de funciones que pueden usarse para efectuar llamadas al sistema. Las hay para distintos lenguajes de programación.

La aplicación llama a una función de la biblioteca de interfaz (mediante una llamada normal) y esa función es la que realmente hace la llamada al sistema.

1.2.4 Interrupciones y excepciones

El SO ocupa una posición intermedia entre los programas de aplicación y el hardware. No se limita a utilizar el hardware a petición de las aplicaciones ya que hay situaciones en las que es el hardware el que necesita que se ejecute código del SO. En tales situaciones el hardware debe poder llamar al sistema, pudiendo deberse estas llamadas a dos condiciones:

- Algún dispositivo de E/S necesita atención.
- Se ha producido una situación de error al intentar ejecutar una instrucción del programa (normalmente de la aplicación).

En ambos casos, la acción realizada no está ordenada por el programa de aplicación, es decir, no figura en el programa.

Según los dos casos anteriores tenemos las interrupciones y las excepciones:

- Interrupción: señal que envía un dispositivo de E/S a la CPU para indicar que la operación de la que se estaba ocupando, ya ha terminado.
- Excepción: una situación de error detectada por la CPU mientras ejecutaba una instrucción, que requiere tratamiento por parte del SO.

1.2.4.1 Tratamiento de las interrupciones

Una interrupción se trata en todo caso, después de terminar la ejecución de la instrucción en curso.

El tratamiento depende de cuál sea el dispositivo de E/S que ha causado la interrupción, ante la cual debe poder identificar el dispositivo que la ha causado.

1.2.4.2 Importancia de las interrupciones

El mecanismo de tratamiento de las interrupciones permite al SO utilizar la CPU en servicio de una aplicación, mientras otra permanece a la espera de que concluya una operación en un dispositivo de E/S.

El hardware se encarga de avisar al SO cuando el dispositivo de E/S ha terminado y el SO puede intervenir entonces, si es conveniente, para hacer que el programa que estaba esperando por el dispositivo, se continúe ejecutando.

En ciertos intervalos de tiempo puede convenir no aceptar señales de interrupción. Por ello las interrupciones pueden inhibirse por programa (aunque esto no deben poder hacerlo las mismas).

1.2.4.3 Excepciones

Cuando la CPU intenta ejecutar una instrucción incorrectamente construida, la unidad de control lanza una excepción para permitir al SO ejecutar el tratamiento adecuado. Al contrario que en una interrupción, la instrucción en curso es abortada. Las excepciones al igual que las interrupciones deben estar identificadas.

1.2.4.4 Clases de excepciones

Las instrucciones de un programa pueden estar mal construidas por diversas razones:

- El código de operación puede ser incorrecto.
- Se intenta realizar alguna operación no definida, como dividir por cero.
- La instrucción puede no estar permitida en el modo de ejecución actual.
- La dirección de algún operando puede ser incorrecta o se intenta violar alguno de sus permisos de uso.

1.2.4.5 Importancia de las excepciones

El mecanismo de tratamiento de las excepciones es esencial para impedir, junto a los modos de ejecución de la CPU y los mecanismos de protección de la memoria, que las aplicaciones realicen operaciones que no les están permitidas. En cualquier caso, el tratamiento específico de una excepción lo realiza el SO.

Como en el caso de las interrupciones, el hardware se limita a dejar el control al SO, y éste es el que trata la situación como convenga.

Es bastante frecuente que el tratamiento de una excepción no retorne al programa que se estaba ejecutando cuando se produjo la excepción, sino que el SO aborte la ejecución de ese programa. Este factor depende de la pericia del programador para controlar la excepción adecuadamente.

1.2.5 Componentes de un sistema operativo

1.2.5.1 Gestión de procesos

Un proceso es simplemente, un programa en ejecución que necesita recursos para realizar su tarea: tiempo de CPU, memoria, archivos y dispositivos de E/S. El SO es el responsable de:

- Crear y destruir los procesos.
- Parar y reanudar los procesos.
- Ofrecer mecanismos para que se comuniquen y sincronicen.

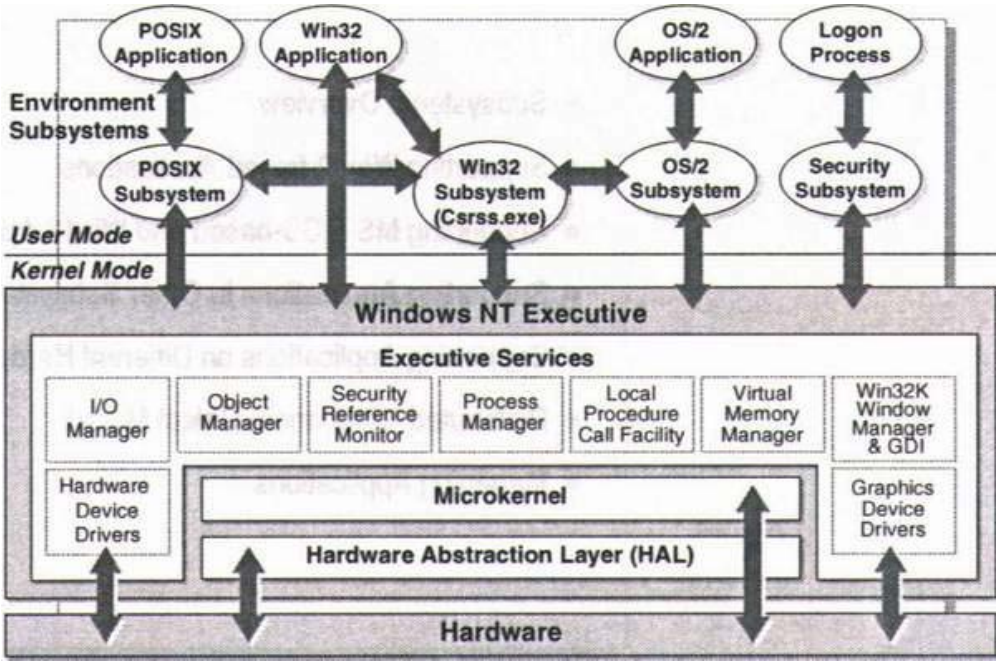


Figura 4. Relación de un Sistema Operativo y el Hardware

1.2.5.2 Gestión de la memoria principal

La memoria es una gran tabla de palabras o bytes que se referencian cada una mediante una dirección única. Este almacén de datos de rápido accesos es compartido por la CPU y los dispositivos de E/S, es volátil y pierde su contenido en los fallos del sistema. El SO es el responsable de:

- Conocer qué partes de la memoria están utilizadas y por quién.
- Decidir qué procesos se cargarán en memoria cuando haya espacio disponible.
- Asignar y reclamar espacio de memoria cuando sea necesario.

1.2.5.3 Gestión del almacenamiento secundario

Un sistema de almacenamiento secundario es necesario, ya que la memoria principal (almacenamiento primario) es volátil y además muy pequeña para almacenar todos los programas y datos. También es necesario mantener los datos que no convenga mantener en la memoria principal. El SO se encarga de:

- Planificar los discos.
- Gestionar el espacio libre.
- Asignar el almacenamiento.

1.2.5.4 El sistema de E/S

Consiste en un sistema de almacenamiento temporal (caché), una interfaz de manejadores de dispositivos y otra para dispositivos concretos. El SO debe:

- Gestionar el almacenamiento temporal de E/S.
- Servir las interrupciones de los dispositivos de E/S.

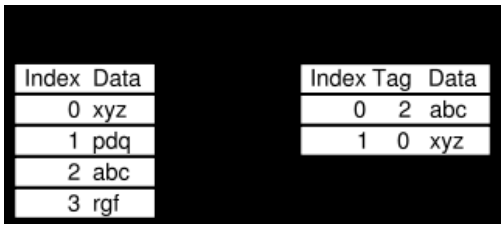


Figura 5. Sistema de memoria

1.2.5.5 Sistema de archivos

Los archivos, ficheros o documentos son colecciones de información relacionada, definidas por sus creadores. Éstos almacenan programas (en código fuente y objeto) y datos tales como imágenes, textos, información de bases de datos, etc... El SO es responsable de:

- Construir y eliminar archivos y directorios.
- Ofrecer funciones para manipular archivos y directorios.
- Establecer la correspondencia entre archivos y unidades de almacenamiento.
- Realizar copias de seguridad de archivos.

Existen diferentes Sistemas de Archivos, es decir, existen diferentes formas de organizar la información que se almacena en las memorias (normalmente discos) de los ordenadores. Por ejemplo, existen los sistemas de archivos FAT, FAT32, EXT2, NTFS...

Desde el punto de vista del usuario estas diferencias pueden parecer insignificantes a primera vista, sin embargo, existen diferencias muy importantes. Por ejemplo, los sistemas de ficheros FAT32 y NTFS, que se utilizan fundamentalmente en sistemas operativos de Microsoft, tienen una gran diferencia para un usuario que utilice una base de datos con bastante información ya que el tamaño máximo de un fichero con un Sistema de Archivos FAT32 está limitado a 4 Gbytes sin embargo en un sistema NTFS el tamaño es considerablemente mayor.

1.2.5.6 Sistemas de protección

Mecanismo que controla el acceso de los programas o los usuarios a los recursos del sistema. El SO se encarga de:

- Distinguir entre uso autorizado y no autorizado.
- Especificar los controles de seguridad a realizar.
- Forzar el uso de estos mecanismos de protección.

1.2.5.7 Sistema de comunicaciones

Para mantener las comunicaciones con otros sistemas es necesario poder controlar el envío y recepción de información a través de las interfaces de red. También hay que crear y mantener puntos de comunicación que sirvan a las aplicaciones para enviar y recibir información, y crear y mantener conexiones virtuales entre aplicaciones que están ejecutándose localmente y otras que lo hacen remotamente.

1.2.5.8 Intérprete de órdenes

El shell del sistema es el principal componente del SO que utiliza el usuario. Este uso se realiza siempre directa o indirectamente a través del intérprete. Generalmente incorpora un lenguaje de programación para automatizar las tareas.

Hay dos tipos de intérpretes de órdenes:

- **Alfanuméricos**: las órdenes se expresan mediante un lenguaje específico usando las cadenas de caracteres introducidas por el terminal.
- **Gráficos**: normalmente las órdenes se especifican por medio de iconos y otros elementos gráficos.

1.2.5.9 Programas de sistema

Son aplicaciones de utilidad que se suministran con el SO pero no forman parte de él. Ofrecen un entorno útil para el desarrollo y ejecución de programas, siendo algunas de las tareas que realizan:

- Manipulación y modificación de archivos.
- Información del estado del sistema.
- Soporte a lenguajes de programación.
- Comunicaciones.

1.2.5.10 Gestor de recursos

Como gestor de recursos, el Sistema Operativo administra

- La CPU o μ-procesador (Unidad Central de Proceso).
- Los dispositivos de E/S (entrada y salida)
- La memoria principal (o de acceso directo).
- Los discos (o memoria secundaria).
- Los procesos (o programas en ejecución).
- ...
- y en general todos los recursos del sistema.

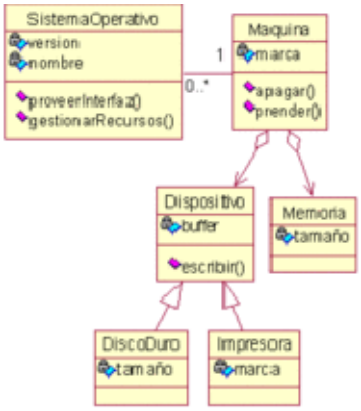


Figura 6. Diagrama de relación de un SO y el hardware

1.2.5.11 Componentes

Los sistemas operativos considerados como un programa han alcanzado un tamaño muy grande, debido a que tienen que hacer muchas tareas. Por esta razón para construir un SO es conveniente dividirlo en componentes más pequeños.

- Gestión de procesos.
- Gestión de memoria.
- Gestión de archivos y directorios.
- Gestión de la E/S (Entrada/Salida).
- Seguridad y protección.
- Comunicación y sincronización entre procesos.
- Intérprete de órdenes

1.2.5.12 Características

- Administración de tareas:
  - Monotarea: Si solamente puede ejecutar un proceso (aparte de los procesos del propio S.O.) en un momento dado. Una vez que empieza a ejecutar un proceso, continuará haciéndolo hasta su finalización o interrupción.
  - Multitarea: Si es capaz de ejecutar varios procesos al mismo tiempo. Este tipo de S.O. normalmente asigna los recursos disponibles (CPU, memoria, periféricos) de forma alternada a los procesos que los solicitan, de manera que el usuario percibe que todos funcionan a la vez, de forma concurrente.

- Administración de usuarios:
  - Monousuario: Si sólo permite ejecutar los programas de un usuario al mismo tiempo.
  - Multiusuario: Si permite que varios usuarios ejecuten simultáneamente sus programas, accediendo a la vez a los recursos del ordenador. Normalmente estos SS.OO. utilizan métodos de protección de datos, de manera que un programa no pueda usar o cambiar los datos de otro usuario.

- Manejo de recursos:
  - Centralizado: Si permite utilizar los recursos de un solo ordenador.
  - Distribuido: Si permite utilizar los recursos (memoria, CPU, disco, periféricos...) de más de un ordenador al mismo tiempo.

1.2.6 Evolución histórica desde las computadoras hasta los actuales Sistemas Operativos

Actualmente el concepto de computadora esta intrínsecamente relacionado al de sistema operativo, aunque éste existe en más aparatos electrónicos aparte de en los ordenadores.

1.2.6.1 Años 40

A finales de los años 1940, con la aparición de la primera generación de computadoras, se realizaba lo que se llama el proceso en serie. Por aquel entonces no existían los sistemas operativos, y los programadores debían interactuar con el hardware del computador sin ayuda externa. Esto hacía que el tiempo de preparación para realizar una tarea fuera excesivo. Además para poder utilizar la computadora debía hacerse por turnos. Para ello se rellenaba un formulario de reserva en el que se indicaba el tiempo que el programador necesitaba para realizar su trabajo. En aquel entonces las computadoras eran máquinas muy costosas lo que hacía que estuvieran muy solicitadas y que sólo pudieran utilizarse en periodos breves de tiempo. Todo se hacia en lenguaje de maquina.

1.2.6.2 Años 50

En los años 50 con el objeto de facilitar la interacción entre persona y computador, los sistemas operativos hacen una aparición discreta y bastante simple, con conceptos tales como el monitor residente, el proceso por lotes y el almacenamiento temporal.

Monitor residente

Su funcionamiento era bastante simple, se limitaba a cargar los programas a memoria, leyéndolos de una cinta o de tarjetas perforadas, y ejecutarlos. El principal problema de estos sistemas era encontrar una forma de optimizar el tiempo entre la retirada de un trabajo y el montaje del siguiente.

Procesamiento por lotes

Como solución para optimizar el tiempo de montaje surgió la idea de agrupar los trabajos en lotes, en una misma cinta o conjunto de tarjetas, de forma que se ejecutaran uno a continuación de otro sin perder apenas tiempo en la transición. Para realizar esto se utilizó una técnica de on-lining. La idea era dedicar un ordenador periférico, de menor coste y potencia, a convertir las tarjetas o la cinta perforada en información sobre cinta magnética, y la salida sobre cinta magnética en salida sobre impresora o cinta perforada. Una vez que se procesaban varios trabajos a cinta, ésta se desmontaba del ordenador periférico, y se llevaba a mano para su procesamiento por el ordenador principal. Cuando el ordenador principal llenaba una cinta de salida, ésta se llevaba al ordenador periférico para su paso a impresora o cinta perforada.

Almacenamiento temporal

Su objetivo era disminuir el tiempo de carga de los programas, simultaneando la carga del programa o la salida de datos con la ejecución de la siguiente tarea. Para ello se utilizaban dos técnicas, el buffering y el spooling.

Sistemas operativos desarrollados

En esta etapa estarían incluidos:

- GM OS: Desarrollado por General Motors para el IBM 701.
- Input Output System: Desarrollado por General Motors y la Fuerza Aérea de los Estados Unidos para el IBM 704.
- FORTRAN Monitor system: Desarrollado por la aviación norteamericana para el IBM 709.
- SAGE (Semi-Automatic Ground Environment): Primer sistema de control en tiempo real, desarrollado para el IBM AN/FSQ7.
- SOS: Desarrollado por el IBM SHARE Users Group para el IBM 709.

1.2.6.3 Años 60

En los años 60 se produjeron cambios notorios en varios campos de la informática, la mayoría orientados a seguir incrementando el potencial de los computadores. Para ello se utilizaban técnicas de lo más diversas:

Multiprogramación

En un sistema multiprogramado la memoria principal alberga a más de un programa de usuario. La CPU ejecuta instrucciones de un programa, cuando el que se encuentra en ejecución realiza una operación de E/S; en lugar de esperar a que termine la operación de E/S, se pasa a ejecutar otro programa. Si éste realiza, a su vez, otra operación de E/S, se mandan las órdenes oportunas al controlador, y pasa a ejecutarse otro. De esta forma es posible, teniendo almacenado un conjunto adecuado de tareas en cada momento, utilizar de manera óptima los recursos disponibles.

Figura 7. Logo de los SO Unix

Tiempo compartido

En este punto se tiene un sistema que hace buen uso de la electrónica disponible, pero adolece de falta de interactividad; para conseguirla debe convertirse en un sistema multiusuario, en el cual existen varios usuarios con un terminal en línea, utilizando el modo de operación de tiempo compartido. En estos sistemas los programas de los distintos usuarios residen en memoria. Al realizar una operación de E/S los programas ceden la CPU a otro programa, al igual que en la multiprogramación. Pero, a diferencia de ésta, cuando un programa lleva cierto tiempo ejecutándose el sistema operativo lo detiene para que se ejecute otra aplicación. Con esto se consigue repartir la CPU por igual entre los programas de los distintos usuarios, y los programas de los usuarios no se sienten demasiado ralentizados por el hecho de que los recursos sean compartidos y aparentemente se ejecutan de manera concurrente.

Tiempo real

Estos sistemas se usan en entornos donde se deben aceptar y procesar en tiempos muy breves un gran número de sucesos, en su mayoría externos al ordenador. Si el sistema no respeta las restricciones de tiempo en las que las operaciones deben entregar su resultado se dice que ha fallado. El tiempo de respuesta a su vez debe servir para resolver el problema o hecho planteado. El procesamiento de archivos se hace de una forma continua, pues se procesa el archivo antes de que entre el siguiente, sus primeros usos fueron y siguen siendo en telecomunicaciones.

Multiprocesador

Permite trabajar con máquinas que poseen más de un microprocesador. En un multiprocesador los procesadores comparten memoria y reloj.

Sistemas operativos desarrollados

Además del Atlas Supervisor y el OS/360, utilizados en máquinas concretas, lo más destacable de la década es el nacimiento de Unix, que hoy en día es una de las plataformas más extendidas en el mundo de la informática.

1.2.6.4 Años 70

Debido al avance de la electrónica, pudieron empezar a crearse circuitos con miles de transistores en un centímetro cuadrado de silicio, lo que llevaría, pocos años después, a producirse los primeros sistemas integrados. Ésta década se podría definir como la de los sistemas de propósito general y en ella se desarrollan tecnologías que se siguen utilizando en la actualidad. Es en los años 70 cuando se produce el boom de los miniordenadores y la informática se acerca al nivel de usuario. En lo relativo a lenguajes de programación, es de señalar la aparición de Pascal y C, el último de los cuales sería reutilizado para reescribir por completo el código del sistema operativo Unix, convirtiéndolo en el primero implementado en un lenguaje de alto nivel. En el campo de la programación lógica se dio a luz la primera implementación de Prolog, y en la revolucionaria orientación a objetos, Smalltalk.

Inconvenientes de los sistemas existentes

Se trataba de sistemas grandes y costosos, pues antes no se había construido nada similar y muchos de los proyectos desarrollados terminaron con costes muy por encima del presupuesto y mucho después de lo que se marcaba como fecha de finalización. Además, aunque formaban una capa entre el hardware y el usuario, éste debía conocer un complejo lenguaje de control para realizar sus trabajos. Otro de los inconvenientes es el gran consumo de recursos que ocasionaban, debido a los grandes espacios de memoria principal y secundaria ocupados, así como el tiempo de procesador consumido. Es por esto que se intentó hacer hincapié en mejorar las técnicas ya existentes de multiprogramación y tiempo compartido.

Características de los nuevos sistemas

Para solventar los problemas antes comentados, se realizó un costosísimo trabajo para interponer una amplia capa de software entre el usuario y la máquina, de forma que el primero no tuviese que conocer ningún detalle de la circuitería.

Sistemas operativos desarrollados

- **MULTICS** (Multiplexed Information and Computing Service): Originalmente era un proyecto cooperativo liderado por Fernando Corbató del MIT, con General Electric y los laboratorios Bell, que comenzó en los 60, pero los laboratorios Bell abandonaron en 1969 para comenzar a crear el sistema UNIX. Se desarrolló inicialmente para el mainframe GE-645, un sistema de 36 bits; después fue soportado por la serie de máquinas Honeywell 6180.

Fue uno de los primeros sistemas operativos de tiempo compartido, que implementó un solo nivel de almacenamiento para el acceso a los datos, desechando la clara distinción entre los ficheros y los procesos en memoria, y uno de los primeros sistemas multiprocesador.

- **MVS** (**M**ultiple **V**irtual **S**torage): Fue el sistema operativo más usado en los modelos de mainframes -ordenadores grandes, potentes y caros usados principalmente por grandes compañías para el procesamiento de grandes cantidades de datos- System/370 y System/390 de IBM, desarrollado también por IBM y

lanzado al mercado por primera vez en 1974. Como características destacables, permitía la ejecución de múltiples tareas, además de que introdujo el concepto de memoria virtual y finalmente añadió la capacidad de que cada programa tuviera su propio espacio de direccionamiento de memoria, de ahí su nombre.

- **CP/M** (**C**ontrol **P**rogram/**M**onitor): Desarrollado por Gary Kildall para el microprocesador 8080/85 de Intel y el Zilog Z80, salió al mercado en 1976, distribuyéndose en disquetes de ocho pulgadas. Fue el SO más usado en las computadoras personales de esta década. Su éxito se debió a que era portable, permitiendo que diferentes programas interactuasen con el hardware de una manera estandarizada. Estaba compuesto de dos subsistemas:

    - **CCP** (**C**omand **C**ontrol **P**rocessor): Intérprete de comandos que permitía introducir los mandatos con sus parámetros separados por espacios. Además, los traducía a instrucciones de alto nivel destinadas a BDOS.

    - **BDOS** (**B**asic **D**isk **O**perating **S**ystem): Traductor de las instrucciones en llamadas a la BIOS.

El hecho de que, años después, IBM eligiera para sus PCs a MS-DOS supuso su mayor fracaso, por lo que acabó desapareciendo.

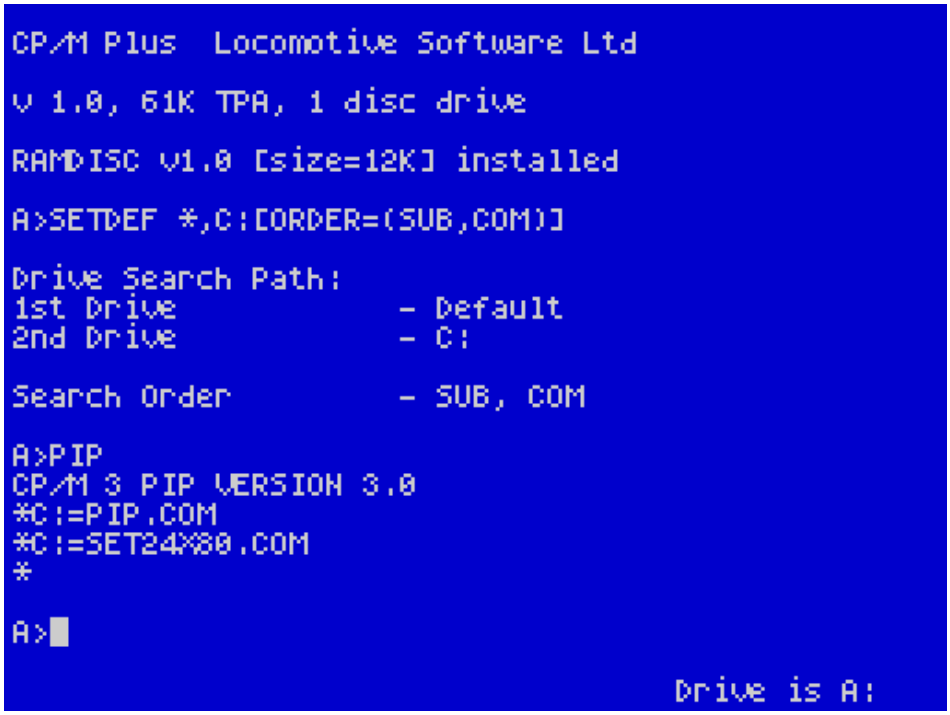


Figura 8. Captura de pantalla del Sistema Operativo CP/M

1.2.6.5 Años 80

Con la creación de los circuitos LSI -integración a gran escala-, chips que contenían miles de transistores en un centímetro cuadrado de silicio, empezó el auge de los ordenadores personales. En éstos se dejó un poco de lado el rendimiento y se buscó más que el sistema operativo fuera amigable, surgiendo menús, e interfaces gráficas. Esto reducía la rapidez de las aplicaciones, pero se volvían más prácticos y simples para los usuarios. En esta época, siguieron utilizándose lenguajes ya existentes, como Smalltalk o C, y nacieron otros nuevos, de los cuales se podrían destacar: C++ y Eiffel dentro del paradigma de la orientación a objetos, y Haskell y Miranda en el campo de la programación declarativa. Un avance importante que se estableció a mediados de la década de 1980 fue el desarrollo de redes de computadoras personales que corrían sistemas operativos en red y sistemas operativos distribuidos. En esta escena, dos sistemas operativos eran los mayoritarios: MS-DOS, escrito por Microsoft para IBM PC y otras computadoras que utilizaban la CPU Intel 8088 y sus sucesores, y UNIX, que dominaba en los ordenadores personales que hacían uso del Motorola 68000.

Apple Macintosh

El lanzamiento oficial se produjo en enero de 1984, al precio de 2495 dólares. Muchos usuarios, al ver que estaba completamente diseñado para funcionar a través de una GUI (Graphic User Interface), acostumbrados a la línea de comandos, lo tacharon de *juguete*. A pesar de todo, el Mac se situó a la cabeza en el mundo de la edición a nivel gráfico.

Figura 9. Logo de los SO Apple

MS-DOS

En 1981 Microsoft compró un sistema operativo llamado QDOS que, tras realizar unas pocas modificaciones, se convirtió en la primera versión de MS-DOS (**M**icro**S**oft **D**isk **O**perating **S**ystem). A partir de aquí se sucedieron una serie de cambios hasta llegar a la versión 7.1, a partir de la cual MS-DOS dejó de existir como tal y se convirtió en una parte integrada del sistema operativo Windows.

Microsoft Windows

Familia de sistemas operativos propietarios desarrollados por la empresa de software Microsoft Corporation, fundada por Bill Gates y Paul Allen. Todos ellos tienen en común el estar basados en una interfaz gráfica de usuario basada en el paradigma de ventanas, de ahí su nombre en inglés. Las versiones de Windows que han aparecido hasta el momento se basan en dos líneas separadas de desarrollo que finalmente convergen en una sola con la llegada de Windows XP. La primera de ellas conformaba la apariencia de un sistema operativo, aunque realmente se ejecutaba sobre MS-DOS.

Actualmente existe Windows Vista.

Figura 10. Logo del SO Microsoft Windows Vista.

1.2.6.6 Años 90

GNU/Linux

En 1991 aparece la primera versión del núcleo de Linux. Creado por Linus Torvalds y un sinfín de colaboradores a través de Internet. Este sistema se basa en Unix, un sistema que en principio trabajaba en modo comandos, estilo MS-DOS. Hoy en día dispone de Ventanas, gracias a un servidor grafico y a gestores de ventanas como KDE, GNOME entre muchos. Recientemente GNU/Linux dispone de un aplicativo que convierte las ventanas en un entorno 3D como por ejemplo Beryl. Lo que permite utilizar linux de una forma muy visual y atractiva.

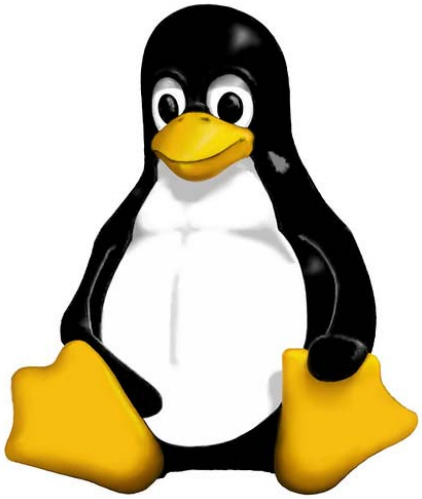


Figura 11. Logo Corporativo del Sistema Operativo Linux.

1.3 El Teléfono Móvil

La telefonía móvil usa ondas de radio para poder ejecutar todas y cada una de las operaciones, ya sea llamar, mandar un mensaje de texto, etc., y esto es producto de lo que sucedió hace algunas décadas.

La comunicación inalámbrica tiene sus raíces en la invención de la radio por Nikola Tesla en los años 1880, aunque formalmente presentada en 1894 por un joven italiano llamado Guglielmo Marconi.

1.3.1 Historia

El teléfono móvil se remonta a los inicios de la Segunda Guerra Mundial, donde ya se veía que era necesaria la comunicación a distancia, es por eso que la compañía Motorola creó un equipo llamado Handie Talkie H12-16, que es un equipo que permite el contacto con las tropas vía ondas de radio que en ese tiempo no superaban más de 600 Khz..

Fue sólo cuestión de tiempo para que las dos tecnologías de Tesla y Marconi se unieran y dieran a la luz la comunicación mediante radio-teléfonos: Martín Coper, pionero y considerado como el padre de la telefonía celular, fabricó el primer radio teléfono entre 1970 y 1973, en Estados Unidos, y en 1979 aparecieron los primeros sistemas a la venta en Tokio (Japón), fabricados por la Compañía NTT. Los países europeos no se quedaron atrás y en 1981 se introdujo en Escandinavia un sistema similar a AMPS (Advanced Mobile Phone System). Y si bien Europa y Asia dieron los primeros pasos, en Estados Unidos, gracias a que la entidad reguladora de ese país adoptó reglas para la creación de un servicio comercial de telefonía celular, en 1983 se puso en operación el primer sistema comercial en la ciudad de Chicago. Este fue el inicio de una de las tecnologías que más avances tiene, aunque continúa en la búsqueda de novedades y mejoras.

Durante ese periodo y 1985 se comenzaron a perfeccionar y amoldar las características de este nuevo sistema revolucionario ya que permitía comunicarse a distancia. Fue así que en los años 1980 se llegó a crear un equipo que ocupaba recursos similares a los Handie Talkie pero que iba destinado a personas que por lo general eran grandes empresarios y debían estar comunicados, es ahí donde se crea el teléfono móvil y marca un hito en la historia de los componentes inalámbricos ya que con este equipo podría hablar a cualquier hora y en cualquier lugar.

Con ese punto de partida, en varios países se diseminó la telefonía celular como una alternativa a la telefonía convencional inalámbrica. La tecnología tuvo gran aceptación, por lo que a los pocos años de implantarse se empezó a saturar el servicio. En ese sentido, hubo la necesidad de desarrollar e implantar otras formas de acceso múltiple al canal y transformar los sistemas analógicos a digitales, con el objeto de darles cabida a más usuarios. Para separar una etapa de la otra, la telefonía celular se ha caracterizado por contar con diferentes generaciones. A continuación, se describe cada una de ellas.

Con el tiempo se fue haciendo más accesible al público la telefonía celular, hasta el punto de que cualquier persona normal pudiese adquirir uno.

Si bien hace un par de años lo genial y primordial de estos aparatos inalámbricos era poder comunicarse, en el año 2001 dio un giro inesperado y se comenzaron a fabricar

los primeros celulares a color, ya no eran esos típicos monocromáticos, ahora poseían una pantalla LCD a colores (al principio fueron de 256 colores y actualmente llegan a los 262.000 y 16.000.000), lo cual impactó a las personas y muchas no dudaron en adquirir uno sin importar cuánto costara. El hecho de que los celulares fueran a color abría un mundo de posibilidades para adaptarles nuevas funciones, como por ejemplo una cámara. Este momento es muy reconocido en la historia de este aparato, ya que junto al "boom" de los celulares a color vino el de los mensajes de texto. Era posible enviar estos usando el teléfono celular, en el cual, con el teclado numérico, se podía escribirlos ahorrándose mucho dinero en vez de hablar, lo que hasta el día de hoy sigue siendo un poco costoso en algunos países.

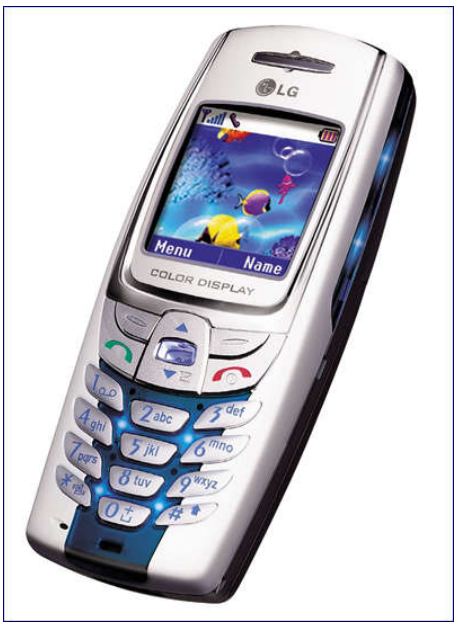


Figura 12. Modelo de un teléfono móvil de la marca LG.

A medida que fue pasando el tiempo los celulares permitían ya no sólo hablar, sino que poder tomar fotos gracias a cámaras que poseían un lente de 3,5 MM (CIF o en algunos casos VGA) y un procesamiento especial de imágenes el cual no comprometía mayores recursos del teléfono. También con este progreso se agregó una característica muy importante que fue la de grabar vídeos y poderlos enviar como Mensaje Multimedia. Si, ya no eran solamente Mensajes de Texto los cuales se podían enviar, sino que ahora eran Mensajes Multimedia de los cuales por las Redes GSM 800/1800/1900 MHz era posible el envío. Una empresa que se caracterizó en esto de los MMS (Mensajes Multimedia) fue Sony Ericsson. Que en este último tiempo han creados teléfonos destinados exclusivamente al uso multimedia, llegando al extremo de poder Ver vídeos, Sacar Fotos, Escuchar Música e Incluso de Jugar con Gráficos 3D, en esta última característica. La compañía que se destacó por crear un centro de juego de bolsillo que pudieses llamar fue Nokia con su modelo N-GAGE, el cual hizo que comenzara a figurar el termino "Memoria RAM", al igual que los computadores estos teléfonos móviles ocupaban Memoria Ram aproximadamente unos 6 a 12 MB, con los cuales era posible escuchar música y jugar con gráficos 3D.

Resumiendo, hace una década aproximadamente los teléfonos celulares se caracterizaban sólo por llamar, pero ha sido tanta la evolución que ya podemos hablar de equipos Multimedia que puede llamar y ejecutar aplicaciones, jugar juegos 3D, ver vídeos, ver televisión y muchas cosas más. Obviamente muchas marcas de placas madres para PC o fabricantes de hardware en general se hacen presentes en los teléfono móviles como por ejemplo: ASUS e INTEL que construyen las placas matrices de lo celulares o ayudan con el acelerador gráfico o el sistema de vídeo. En fin, debemos tener conciencia y prepararnos para lo que se viene más adelante y pensar que el teléfono celular ya no es tan sólo para hablar.

1.3.2 Generaciones

1.3.2.1 1ª Generación.

En 1979, se dio en los países asiáticos el nacimiento de la primera generación de celulares, con tecnología analógica que utiliza ondas de radio para transmitir una comunicación: la voz se transmite sin ningún tipo de codificación. Los móviles eran muy pesados y de gran tamaño, debido a que tenían que realizar una emisión de gran potencia para poder lograr una comunicación sin cortes ni interferencias. Los enlaces tenía una velocidad de 2400 baudios, y la seguridad era muy baja o no existía. La tecnología predominante en esta generación fue la AMPS Advanced Mobile Phone System.

Figura 13. Imagen de un teléfono móvil de primera generación.

1.3.2.2 2ª Generación.

En Europa en la Segunda Generación de Celulares la diferencia primordial con la anterior es que se utiliza Tecnología Digital, los protocolos de comunicación son mucho más sofisticados como GSM (Global System for Mobile communication), IS – 136 (TIA/EIA 136 O ANSI 136) y CDMA (Code Division Multiple Access) y por último PDC (Personal Digital Comunication), entre otros. La velocidad en ésta es mucho más alta para la voz, aunque se tenía problemas en los datos, ya que se incorporaron servicios como

FAX, SMS, así como los servicios de WAP (Wireless Access Protocol). Esto último se debió a que el uso de Internet crecía de manera progresiva. WAP es lento y pesado, cosa que se fue mejorando al punto de que se pudo desarrollar aplicaciones para los equipos 2G, por ejemplo las descargas JAVA2ME. Una de las bondades de esta tecnología es el cifrado de datos y voz para que ésta sólo fuese descifrada por el celular receptor de destino.

Figura 14. Teléfono móvil 2G

1.3.2.3 2'5ª Generación

Dado que la tecnología de 2G fue incrementada, se puede incluir dentro de la 2.5 en la cual se incluyen nuevos servicios como EMS y MMS, pero con muchas diferencias:

-EMS es el servicio de mensajería mejorado, permite la inclusión de melodías e iconos dentro del mensaje basándose en los sms; 1 EMS equivale a 3 o 4 sms.

-MMS (Sistema de Mensajería Multimedia) Este tipo de mensajes se envían mediante GPRS y permite la inserción de imágenes (jpeg, gif, bmp), sonidos (amr, midi), videos (.3gp) y texto. Un MMS se envía en forma de diapositiva, en la cual cada plantilla solo puede contener un archivo de cada tipo aceptado, es decir, solo puede contener una imagen, un sonido y un texto en cada plantilla, si de desea agregar mas de estos tendría que agregarse otra plantilla. Cabe mencionar que no es posible enviar un vídeo de más de 15 segundos de duración.

-GPRS y IP-GPRS, que es un servicio para enviar y recibir "paquetes" de datos a altas velocidades debido a que se subdividen y comprimen y son enviados a intervalos regulares: “conmutación de paquetes”. La conexión se usa al momento de utilizar el canal, por ello la facturación es por tamaño de datos.

Figura 15. Teléfono móvil 2´5G

1.3.2.4 3ª Generación.

En 2001 se lanza en Japón la 3G de celulares, los cuales están basados en los UMTS (servicios General de Telecomunicaciones Móviles). En este caso se dio uno de los pasos finales en lo que es la telefonía móvil y la Informática. La novedad más significativa fue la incorporación de una segunda cámara para realizar videollamadas, es decir hablar con una persona y verla al mismo tiempo por medio del teléfono móvil.

Un vistazo a los principales sistemas operativos para dispositivos móviles no tiene que ver con cámaras, conexiones ni otra parafernalia. El coeficiente intelectual de los PDA (asistentes digitales) y teléfonos inteligentes (SmartPhones) está determinado exclusivamente por el sistema operativo elegido por sus fabricantes. En 2008 se venderán más de 800 millones de PDA, de los que 250 serán SmartPhones.

Figuras 16 y 17. Teléfonos móviles 3G (Nokia N95 e Iphone de Apple)

1.4 Sistemas Operativos para móviles

La evolución del hardware de los terminales móviles caería en saco roto si no hubiese una renovación constante del software que los gobierna.

El reciente lanzamiento de Windows Vista ha vuelto a poner de manifiesto la necesidad de actualizar de forma periódica los sistemas operativos para que los ordenadores sigan satisfaciendo las cambiantes necesidades de los usuarios. Con los dispositivos portátiles en general y los SmartPhones en particular sucede exactamente lo mismo.

Habitualmente, el protagonismo en este ámbito lo suele acaparar el hardware: cámaras de hasta 5 Megapíxeles, pantallas táctiles de gran tamaño, baterías de gran autonomía, teclados QWERTY que potencian el uso del terminal más allá del simple teclado numérico, etc. Pero, detrás de estas características se encuentra el sistema operativo, el verdadero artífice de todo lo que hace el terminal. Pese a haber dos opciones en evidente liza, Windows Mobile y Symbian, también han aparecido otras propuestas capaces de poner en jaque a estas dos veteranas: Linux, Java e incluso Mac OS X con la llegada del iPhone.

1.4.1 Blackberry

Research in motion, la empresa que creó esta exitosa herramienta de correo push, dejó muy claras desde un principio sus intenciones con la familia de soluciones Blackberry: conquistar a todos los profesionales y usuarios que desean estar permanentemente conectados a su correo electrónico. Estos equipos están gobernados por un sistema operativo desarrollado en J2ME, un lenguaje conocido también como Java Micro Edition. Su semejanza con C le ha granjeado el apoyo de los desarrolladores, que han visto en esta plataforma una base interesante para sus aplicaciones.

El sistema operativo de Blackberry se parece más a Garnet OS y Palm OS que a Windows Mobile, especialmente en todo lo relacionado con el trabajo ofimático y la gestión del correo, ámbitos en los que se desenvuelve a las mil maravillas. Al tiempo, juega su gran baza en cuestiones de seguridad. Por el contrario, no está tan bien dotado en lo que concierne a la reproducción de archivos multimedia como los terminales que recurren a Windows o Symbian.

Figura 18. Sistema Blackberry

1.4.2 Linux

Aunque hasta la fecha se han impuesto los terminales gobernados por Symbian y Windows Mobile, también hay un hueco para las soluciones que recurren al software de libre distribución. En esta tesitura, destaca Qtopia, un sistema operativo basado en Linux bajo licencia GPL (aunque existe una versión comercial). Esta plataforma para dispositivos móviles gobernó algunos reproductores multimedia portátiles (PMP) antiguos de Archos. Su sencilla interfaz gráfica y la posibilidad de integrarla en dispositivos con pantalla táctil la convierten en una alternativa excelente para SmartPhones, aunque por desgracia son pocos los fabricantes que se aventuran a apostar por Linux en sus soluciones.

Entre sus bondades destacan sus escasas exigencias en lo que concierne al hardware y sus amplias posibilidades (soporte wireless, Java, PIM, multimedia, etc.). Por esta razón, gobierna muchos dispositivos de la familia Zaurus de Sharp. Además, dentro de poco podremos ver teléfonos móviles creados a partir de este sistema, como Greenphone de TrollTech. Por su parte, Motorola, que ya apostó por Linux en algunos de sus modelos a pesar de que apenas hicieron «ruido» en nuestro mercado, es también noticia porque a lo largo de este año presentará dispositivos como ROKR2, una versión renovada del famoso Motorola con iTunes.

Figura 19. Imagen de una pantalla de un teléfono móvil utilizando SO Linux

1.4.3 Mac OS X

La empresa de la manzana ha tratado de dar una nueva vuelta de tuerca a un mercado en el que se presenta como «novata». Y es que iPhone promete ser un teléfono muy atractivo por su sofisticado diseño que, además, incorporará el navegador Safari, iPhoto, Coverflow y los famosos widgets, prestaciones que probablemente eclipsarán la ausencia de otras características que muchos usuarios echarán de menos, como el soporte UMTS. Apple ha adecuado uno de sus sistemas operativos de uso actual para el iPhone, en concreto Mac OS X v10.4 denominado Tiger, siguiendo con la costumbre de Jobs de ponerle a sus sistemas operativos nombres de grandes felinos (recordemos que los anteriores OS de Apple se llamaban Cheetah, Puma, Jaguar, Panther...).

Tiger fue lanzado al público el día 29 de Abril de 2005 como el sucesor de Mac OS X v10.3 denominado "Panther", el cual fue lanzado 18 meses antes. Algunas de las nuevas características incluyen un sistema de búsqueda rápido denominado Spotlight, una nueva versión de navegador web Safari, Dashboard, un nuevo tema gráfico unificado y un mejorado soporte para 64 bits de las Power Mac G5. Tiger es también la primera versión de un sistema operativo de Apple que trabaja en la plataforma Intel x86, aunque también está pensado para que trabaje en arquitectura Apple-Intel solamente como las MacBook Pro, MacBook, Mac Mini Intel y Mac Pro. En Octubre de 2007 saldrá la nueva versión de un sistema operativo Apple, este será Mac Os X v10.5 denominado Leopard.

Figura 20. Imagen del Sistema Operativo Mac OS X sobre un teléfono móvil.

1.4.4 Windows Mobile

El líder indiscutible en el sector del PC impone también su ley en el mundo de los dispositivos móviles con una versión reducida de Windows basada en Win32. No obstante, esta solución es el resultado de la evolución de los ya algo vetustos Windows CE y Pocket PC. Su interfaz guarda cierta similitud con la de un Windows tradicional. Por el momento, Mobile 5 es uno de los desarrollos más completos para gadgets de bolsillo con soporte para redes UMTS, algo que no acaban de aceptar los muchos detractores de los de Redmond. Entre sus ventajas destaca la gran cantidad de software suministrado junto al sistema operativo, como la versión Pocket de Microsoft Office, con Word, Excel e incluso PowerPoint. Además, es más fácil desarrollar para esta plataforma que, por ejemplo, para Symbian, ya que los entornos de programación son prácticamente idénticos a los empleados en PC. Por su parte, destaca también su versatilidad en el escenario multimedia en gran medida gracias a la integración de Windows Media Player 9, un reproductor capaz de leer una gran cantidad de formatos de audio y vídeo que abarcan desde el WMV hasta AVI.

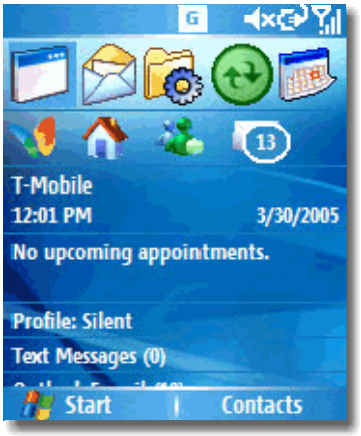


Figura 21. Sistema Operativo Windows Mobile.

Esta plataforma ha sido respaldada por firmas como HTC, HP y Palm, que decidió «pasarse» a este sistema en sus últimos dispositivos de la gama Treo.

1.4.5 Palm OS

Esta plataforma fue creada en un principio para gobernar agendas digitales sencillas, pero su constante evolución le ha permitido competir directamente con gigantes como Windows Mobile o Symbian. Actualmente, el nombre de Palm OS ha sido reemplazado por Garnet OS debido a que Access, la empresa propietaria de la antigua PalmSource, tiene la intención de dar un nuevo «aire» a su producto estrella.
Una de las principales tantos de Palm Os es su intuitiva y ligera interfaz, que repercute directamente en lo poco exigente que esta plataforma es con el hardware (funciona con apenas 300 Kbytes de memoria RAM). Algunas tareas, como eliminar una aplicación, son tan sencillas que no es preciso emplear un gestor o asistente. Por el contrario, en el ámbito multimedia no puede competir con otras opciones, como Windows Mobile y su reproductor Media Player.
En la actualidad, el futuro de este desarrollo es incierto, pero todo parece indicar que los ingenieros de Access están trabajando en una nueva plataforma móvil basada en Linux que atiende al nombre de Linux Platform. Por esta razón, quizás muy pronto tengamos un nuevo competidor para Windows Mobile y sus gadgets.

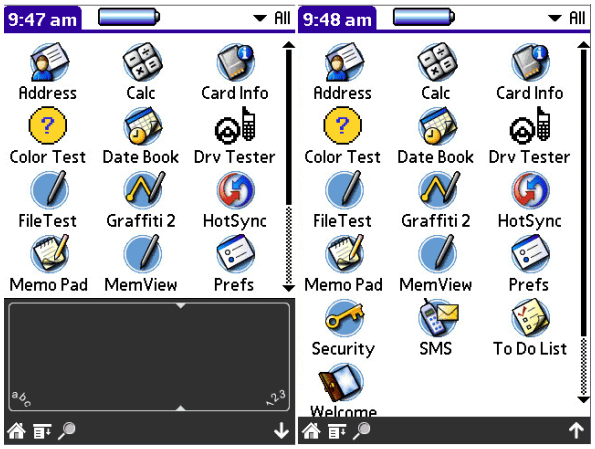


Figura 22. Sistema Operativo PalmOS

1.4.6 Symbian

Este es, sin duda, uno de los sistemas operativos más utilizados en los teléfonos correctamente apellidados «inteligentes». Su origen se remonta al nacimiento de los miniPC de Psion, de manera que ya desde el principio apuntaba buenas cualidades para plataformas móviles. Su éxito está ligado al de los terminales de la serie S60 de Nokia (de los que se han vendido más de 50 millones en todo el mundo). No obstante, los S60 no son los únicos terminales que utilizan Symbian, ya que en los teléfonos más avanzados de Sony Ericsson (como el P990) encontramos una versión de esta plataforma denominada UIQ, que permite utilizar pantallas táctiles, una prestación presente también en la serie S90 de Nokia. Por su parte, la serie S80 está principalmente asociada a los famosos dispositivos Communicator. En este caso, la pantalla no es táctil, pero cuenta con grandes posibilidades para el trabajo en oficina y demás entornos profesionales.

Entre las principales cualidades de Symbian frente a otras opciones destacan el buen uso que hace de la memoria principal, que apenas consume batería y, además, que se trata de un estándar abierto. Gracias precisamente a esta última característica los usuarios pueden instalar software y desarrollarlo con facilidad empleando múltiples lenguajes, como Java, C++, Flash y Perl.

Figura 23. Sistema Operativo Symbian

1.5 Sony Ericsson P800

El Sony Ericsson P800 fue toda una revolución tecnológica haya por el año 2002, un móvil que incorporaba Symbian 7.0 perteneciente a la serie UIQ. Este teléfono es GSM tribanda (900/1800/1900) soporta varios formatos de wap además de tener email integrado (en sus formatos pop3, smtp, imap3 y mime). Tiene una memoria interna de solo lectura de 40 megabytes y otra libre de 12 megabytes, se le pueden incorporar tarjetas de memoria del tipo Memory Stick Duo desde 8 megas hasta 128. Además incorpora una batería de litio, pantalla TFT, reproductor de audio y video, cámara de fotos y bueno un sinfín de utilidades que en su momento fueron de lo mas puntero. Pero lo más interesante del mismo es como se ha comentado Symbian, el cual soporta varios lenguajes de programación: C++, PersonalJava (pJava), J2ME MIDP y Visual Basic. A partir de estos se van a

desarrollar las herramientas de las que antes se ha hablado, intentando aprovechar todas las capacidades que brinda este terminal.

Figura 24. Imagen del teléfono móvil P800 de Sony Ericsson

1.6 Objetivo del proyecto

El objetivo de este proyecto ha sido intentar aglutinar de una manera sencilla y totalmente comprensible las distintas capacidades de Symbian OS, desarrollando un conjunto de herramientas didácticas acompañadas por unas prácticas para facilitar la labor de todas aquellas personas que desean aprender a utilizar este sistema. Elaborando un dossier que permite agilizar el aprendizaje y manejo de uno de los principales sistemas operativos existentes para terminales móviles.

Se intenta suavizar la toma de contacto con una tecnología tan novedosa y que en un principio por falta de información puede resultar bastante caótica. Se han intentado incluir todas las tecnologías móviles actuales dando una visión objetiva de cada una y desarrollando las que parece que pueden ofrecer mayor número de posibilidades.

Capitulo 2

Symbian OS

2.1 Symbian OS

Millones de personas alrededor del mundo usan teléfonos celulares para comunicarse dentro o fuera de su país. Frescos aún en nuestra memoria, están esos teléfonos móviles con pantallas de calculadora de los años 70: números conformados por focos rojos, amarillos o verdes, y posteriormente, por pantallas de cristal líquido monocromáticas, todos ellos compartiendo la misma función esencial: hacer y recibir llamadas telefónicas.

Sin embargo, los aparatos de recientes generaciones tienen más servicios y aplicaciones: fotografía, video, juegos multimedia, envío y recepción de correos electrónicos o mensajes instantáneos, navegación en Internet, etc. Poco a poco, el teléfono celular se ha transformado de un dispositivo de comunicación auditiva básica a virtuales centros de entretenimiento y asistencia de información personal.

Un teléfono es una radio, muy sofisticada, pero en esencia una radio, como un sintonizador de AM/FM. Si bien es conocido, el teléfono fue inventado en 1876 por Alejandro Graham Bell, y las comunicaciones sin cables o Wireless fueron desarrolladas por Nikolai Tesla en 1880 y formalmente presentadas, por el italiano Guglielmo Marconi en 1894; ha sido en los últimos 20 años, cuando estas tecnologías se integraron en esos simpáticos aparatos que portamos en el cinturón.

Para lograr las nuevas funcionalidades que tiene un teléfono celular se necesita cierta inteligencia, además de que es indispensable un sistema operativo. Symbian es uno de los más importantes y funcionales, aunque existen otros como Palm® que funciona en asistentes digitales personales (PDA, por sus siglas en inglés), los cuales se hicieron populares por su interfaz que utiliza una pluma electrónica y reconoce la escritura a mano en su pantalla; o Windows CE de Microsoft®, una variante del sistema operativo Windows, que se integra a diversos dispositivos electrónicos con mínima capacidad de almacenamiento, incluyendo PDAs y teléfonos móviles.

Symbian fue diseñado para integrarse fácilmente, con el software y el hardware en dispositivos móviles. Los sistemas que utilizan Symbian pueden ser divididos en cuatro categorías: serie 60, para teléfonos que el usuario maneja con una sola mano; serie 80, para teléfonos con pantalla horizontal grande y teclado; serie UIQ, para teléfonos que utilizan una pluma electrónica similar a los asistentes digitales; y otros, para futuros desarrollos o que no están en las categorías anteriores.

Symbian es producto del trabajo conjunto de varias empresas líderes en telefonía celular, como Nokia®, Ericsson®, Motorola® y Psion®, quienes fundaron esta empresa en 1998 con el objeto de desarrollar un sistema operativo abierto para las diversas plataformas de teléfonos móviles. En 2003 Motorola vendió el 13% de su participación a Nokia, que se hizo con el 32.2% de la compañía.

El objetivo de Symbian fue crear un sistema operativo para terminales móviles que pudiera competir con el de Palm o el Smartphone de Microsoft.

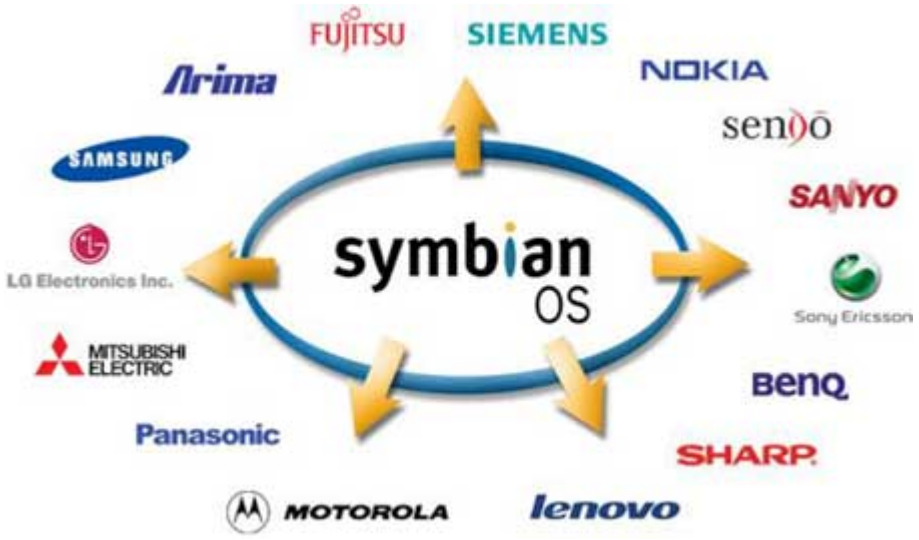


Figura 25. Consorcio de Compañías asociadas bajo Symbian

A diferencia de cualquier computadora, la programación de los teléfonos celulares tiene sus limitaciones. Es por esto que Symbian está diseñado para residir en un espacio muy pequeño, hacer un uso dinámico de escasos recursos de memoria, administrar eficientemente la energía y soportar en tiempo real los protocolos de comunicación y telefonía, además de ser más "gentil" con el usuario y tolerante a fallos, en comparación con un sistema operativo de PC. Por ejemplo: cuando se le quita la batería al teléfono mientras está encendido, el usuario esperaría que su información se encuentre íntegra al volver a activarlo, y no frustrarse por el hecho de que sus datos guardados se hayan perdido o peor: que el teléfono deje de funcionar, porque el sistema operativo se dañó cuando ocurrió el corte de energía.

Técnicamente, el sistema operativo Symbian es una colección compacta de código ejecutable y varios archivos, la mayoría de ellos son bibliotecas vinculadas dinámicamente (DLL por sus siglas en inglés) y otros datos requeridos, incluyendo archivos de configuración, de imágenes y de tipografía, entre otros recursos residentes. Symbian se almacena, generalmente, en un circuito flash dentro del dispositivo móvil. Gracias a este tipo de tecnología, se puede conservar información aun si el sistema no posee carga eléctrica en la batería, además de que le es factible reprogramarse, sin necesidad de separarla de los demás circuitos.

Las aplicaciones compatibles con Symbian se desarrollan a partir de lenguajes de programación orientados a objetos como C++, Java (con sus variantes como PJava, J2ME, etc.), Visual Basic para dispositivos móviles, entre otros, incluyendo algunos lenguajes disponibles en versión libre. La extensa variedad de aplicaciones que se pueden desarrollar van desde administradores de archivos hasta visualizadores de películas, guías de ciudades, mapas, diccionarios, emuladores de juegos, por mencionar algunas. Cada aplicación se puede instalar en el teléfono con la ayuda de una computadora, una interfaz USB o firewire, dependiendo del modelo de teléfono y el cable correspondiente. Las aplicaciones

se graban en la memoria flash del teléfono dentro del proceso de sincronización de archivos, contactos y correos electrónicos.

Symbian es actualizable. Esta tarea puede realizarla el usuario, dependiendo del modelo de su equipo, a través de los sitios en Internet de los fabricantes de teléfonos o bien, al obtener el disco compacto o tarjeta de memoria flash de los distribuidores autorizados. Adicionalmente, Symbian proporciona una interfaz gráfica fácil de comprender, llena de íconos y opciones, con lo cual se evita que el usuario deba aprender manuales inmensos para explotar las capacidades de su equipo de comunicación móvil.

Paradójicamente, Symbian también es vulnerable a los virus que afectan a computadoras personales y asistentes digitales (PDA). La manera más común de contagio es cuando el aparato está en comunicación con algún otro dispositivo contaminado en la red local (si el teléfono es compatible con la norma IEEE 802.11) o en la red personal (si es compatible con Bluetooth). Menos frecuente es la infección por mensajes cortos (MMS); sin embargo, estos virus pueden bloquear aplicaciones e incluso el sistema de archivos, no obstante, también hay métodos para prevenirlos y eliminarlos instalando programas antivirus.

Finalmente, Symbian contiene una muy variada y extensa colección de bibliotecas para implementar muchos de los estándares de la industria de teléfonos de generación 2, 2.5 y 3. Symbian en un sistema muy completo, diseñado prácticamente para cualquier dispositivo móvil. Además de la gran comunidad de desarrolladores que trabajan en este sistema, su plataforma abierta permite la instalación de una gran variedad de programas que poco a poco mejorarán su funcionamiento, a favor de la versatilidad de las comunicaciones personales.

2.2 Principales características de Symbian OS

Symbian posee ciertas características que influyen de manera determinante en el desarrollo de aplicaciones. Primero, Symbian es un sistema operativo basado en ROM, no siempre ha habido la posibilidad de grabar datos en la memoria del teléfono, aunque ahora generalmente se disponga de memorias flash. Segundo, ha sido diseñado para trabajar en dispositivos con una limitada potencia de procesamiento y recursos.

Symbian esta basado en un microprocesador kernel. Una mínima porción del sistema tiene privilegios del kernel, el resto se ejecuta con privilegios de usuario. Una de las tareas del kernel es la de manejar las interrupciones y prioridades. En Symbian, cada aplicación corre en sus propios procesos y tiene acceso solo a su propio espacio de memoria. Este diseño hace que la aplicaciones para Symbian sean orientadas a “single threads” y no múltiples.

Pero Symbian, también posee componentes que periten el diseño de aplicaciones multiplataforma, dando la posibilidad de diferentes tamaños de pantalla, color, resolución, teclados, etc… La mayoría de estos componentes han sido diseñados en C++.

Todas estas características permiten que los dispositivos con Symbian puedan estar en constante funcionamiento sin necesidad de ser reseteados, preservando, de esta forma, la información del usuario.

Symbian OS incluye el *kernel*, los drivers para el hardware y ciertos componentes del *middleware*. Symbian entrega el código del sistema operativo a los fabricantes que con vistas a ser modificado. El *kernel* de Symbian está considerado como uno de los más

potentes para dispositivos móviles. Encima del *kernel* se encuentran las librerías que soportan los gráficos, sistema de archivos, memoria, funciones de telefonía, conectividad y gestión de aplicaciones. La capa de *middleware* incluye las librerías básicas de interfaz de usuario, codecs, soporte Java y soporte de protocolos de mensajería y navegación.

Con motivo de favorecer la diferenciación entre fabricantes, el interfaz de usuario está totalmente separado del resto del sistema operativo. Actualmente, el marco del IU, ciertos componentes relacionados del middleware, el entorno de ejecución para aplicaciones, así como la familia de aplicaciones son proporcionados por S60, UIQ o MOAP. El entorno de desarrollo sin embargo está unificado para S60 y UIQ bajo el entorno Carbide C++ de Nokia (basado en Eclipse).

Para correr Symbian OS se necesita como mínimo un procesador ARM9 con una unidad de gestión de memoria. Actualmente, el sistema operativo ha sido portado a diferentes arquitecturas de microprocesador incluyendo Freescale, Texas Instruments y ST Microelectronics.

Como software de sistema operativo, Symbian proporciona las rutinas y los servicios subyacentes para las aplicaciones. Técnicamente el sistema operativo Symbian es una colección compacta de código ejecutable y varias archivos, la mayoría de ellos bibliotecas DLL (aunque también encontramos archivos de configuración, tipografías, imágenes y otros recursos).

Figura 26. Dispositivo móvil utilizando GPS

Por norma general, el sistema operativo Symbian lo encontramos cargado en la memoria flash del teléfono móvil, de esta forma podemos conservar el sistema operativo aun cuando no tengamos batería. Además, el estar dispuesto en una memoria aparte, facilita su re programación o actualización sin necesidad de separarla de los demás circuitos.

El desarrollo de aplicaciones para Symbian es sencillo ya que no es necesario aprender un lenguaje de programación nuevo, se puede programar a partir de lenguajes de programación de pc como Java, C++ Visual Basic, Python, Perl, Flash Lite (entre otros). Este hecho ha conseguido que existan en la actualidad millones de aplicaciones para

móviles Symbian que realicen todas las tareas imaginables: juegos, mapas, guías de ciudades, reproductores de vídeo, traductores, diccionarios, administradores de archivos, emuladores de otros dispositivos como consolas, navegadores web

2.3 Evolución del Sistema Operativo Symbian

Aunque su nombre ha cobrado verdadera fuerza sólo en los últimos años, el pasado de este sistema operativo está profundamente enraizado con la evolución de los dispositivos móviles. Es más: fue su compañía madre, la británica Psion, quien definió las características de un PDA allá por 1984, lanzando el primer organizador digital fabricado a gran escala.

En 1989 la empresa comenzó a desarrollar EPOC, uno de los primeros sistemas operativos para PDA. Sin embargo -y aún cuando el programa llegó a sumar cinco versiones, Psion fue incapaz de capitalizar su éxito y para 1997 sus planes para licenciarlo se iban por la borda, con apenas tres clientes y menos de mil unidades vendidas.

En junio del año siguiente, Psion vendió su división de software a una coalición de fabricantes de teléfonos móviles. De esa forma, la versión 6 de EPOC tomó el nombre de su nueva empresa: Symbian.

En la actualidad, Symbian es una inversión conjunta de Ericsson, Panasonic, Siemens AG, Samsung y Sony Ericsson - quienes promedian alrededor del 10% cada uno - más la participación mayoritaria de Nokia, con casi el 48% de la propiedad. Todavía basada en el Reino Unido, la empresa tiene una plantilla de 750 empleados, trabajando en un sistema que es utilizado por 15 licenciatarios en todo el mundo.

Altamente flexible, Symbian es un sistema disponible en cuatro versiones: UIQ, Nokia Series 60, 80, 90 y Nokia 9200 Communicator, cuyas características dependen directamente del dispositivo sobre el cual va a funcionar (si es un teléfono móvil o PDA, si tendrá lápiz o teclado, etcétera).

Figura 28. Logotipo del sistema operativo Symbian

2.3.1 Versiones Symbian OS

EPOC 16: Psion lanza entre los años 1991 y 1998 su Serie 3 que se compone de varios dispositivos, estos usaban el sistema operativo EPOC16 más conocido como SIBO

EPOC32: Los dispositivos de la llamada Serie 5, lanzado en 1997 utilizaba la primera iteración del sistema operativo EPOC32.

EPOC4: Oregon Osaris y Geofox 1 fueron lanzados usando ER4.
En 1998, Symbian Ltd. fue formada como sociedad entre Ericsson, Nokia, Motorola y Psion, para explorar la convergencia entre PDA y los teléfonos móviles.

EPOC5: En el año 1999 sale a la luz un nuevo sistema operativo creado por Psion, este es llamado EPOC5 y será el antecesor de Symbian. Contaba, entre otras, con las siguientes características: Java, color, mejoras en general de las versiones previas de EPOC e interfaz gráfica basada en EIKON.

Symbian OS 5: En 1999 Psion y Ericsson lanzaron sus primeros aparatos con un sistema Symbian, estos eran: Psion Series 5mx, Series7, Psion Revo, Psion Netbook y Ericsson MC218, usaban ER5.

Symbian OS 5.1: En el año 2000 Ericsson lanza su primer teléfono utilizando ER5, este era el Ericsson R380. No era un teléfono "abierto" ya que no se le podía instalar software. Psion creo un gran número de prototipos para la siguiente generación de PDA´s que nunca llegaron al mercado, incluyendo el sucesor de Bluetooth Revo conocido como Conan, estos utilizaban ER5u. La "u" en el nombre de este sistema se refiere a que utilizaba Unicode.

Symbian OS v6.0: También llamado ER6, su lanzamiento fue en el año 2000. Nokia lanza con esta versión su primer teléfono móvil Symbian llamado Nokia 9210. Este incluye nuevas características como teclado qwerty, una interfaz gráfica mas completa, Wap y mejoras en Java. También incluye servicios de GSM, voz, mensajes, email e internet, todo esto integrado conjuntamente con aplicaciones de uso diario, como agenda y contactos. Las opciones de conectividad incluyen sincronización con el pc, transferencia de ficheros e infrarrojos.

Symbian OS v6.1: En el año 2001 se hace una corrección del anterior Symbian, se arreglan fallos y se añade Bluetooth, mensajes multimedia, mayor seguridad, varias vistas en las aplicaciones, soporte de clipboard para multimedia y algunos elementos mas de interfaz de usuario.

Symbian OS 7.0: Esta es una versión muy importante de Symbian, ya que en el año 2003 (año de lanzamiento de esta nueva versión) aparecen las distintas interfaces que utilizamos hoy en día:

- **UIQ:** Sony Ericsson P800, P900, P910, Motorola A925, A1000.
- **Series 60:** Nokia 3230, 6600, 7310.
- **Series 80:** Nokia 9300, 9500.
- **Series 90:** Nokia 7710

También se produce la aparición de los teléfonos FOMA en Japón utilizando este sistema.

Symbian OS 7.0s: Pequeña mejora de la versión 7.0, incluye algunos lenguajes que faltaban y proporciona una mayor compatibilidad con Symbian 6.x y anteriores, esta mejora esta originada en gran parte para proporcionar compatibilidad entre el Nokia 9500 Comunicator y el Nokia 9210 Comunicator. En 2004 Psion vende Symbian a una conjunción de compañías y nace el primer virus para Symbian, este se llamo Cabir y era un gusano que se transmitía a través del Bluetooth.

Symbian OS 8.0: Proviene del año 2004, incluye las últimas versiones de Java, un nuevo sistema de sincronización y aceleración por Hardware (MDF). Existen dos versiones de la misma, por un lado la 8.0a, que usa el kernel de versiones anteriores tales como 6.1, 7.0 y 7.0s (el kernel que utiliza es el EKA1), esta permitía compatibilidad total con anteriores modelos. Por otro lado esta la versión 8.0b, que usa un nuevo kernel (EKA2) a tiempo real. Los kernel se comportan de manera similar del lado del usuario, pero internamente son muy diferentes. El kernel EKA2 permite a Symbian actuar directamente sobre el teléfono. Actualmente los teléfonos tienen dos microprocesadores, uno para el sistema operativo del fabricante del teléfono que controla las llamadas, mensajes y comunicaciones en general, y otro para Symbian.

Symbian OS 8.1: Es básicamente una reestructuración de la versión 8.0, disponible también para los dos tipos de kernel EKA1 y EKA2 en sus versiones 8.1a y 8.1b respectivamente. La versión 8.1b, con el kernel EKA2 y un único chip fue muy popular entre las compañías japonesas que deseaban la ayuda en tiempo real pero no permitir la instalación de aplicaciones abiertas.

Symbian OS 9.0: Esta versión solo fue utilizada internamente por Symbian, marcó el final del camino de Eka1 8.1a (esta fue la última versión Symbian de este kernel). En esta versión los cambios son substanciales, sobre todo en las herramientas y en la seguridad. Pasaron de utilizar ARM-4 a ARM-5 sin perder compatibilidad con las versiones anteriores.

Figuras 29 y 30. Captura de pantalla de un juego de billar en Symbian OS versión 9.1

Symbian OS 9.1: Lanzado a principios del año 2005, incluye nuevas características relacionadas con la seguridad, particularmente un modulo polémico de la plataforma que obliga a que todo código vaya firmado para poder ser instalado. Hubo una gran polémica por este modulo ya que mientras Symbian defendía que gracias a el mismo se mantenía una seguridad total, algunas voces se alzaron reivindicando una mayor libertad para el usuario pidiendo el derecho de poder instalar lo que quisieran en sus terminales. La tercera edición de la plataforma S60 tiene Symbian OS 9.1 como por ejemplo el M600 o el P990 de Sony Ericsson y también algunos modelos de la serie N de Nokia como por ejemplo el N80 o el N73.

Symbian OS 9.2: Versión perteneciente al primer cuatrimestre del año 2006. Incluyen ayuda para la gerencia del dispositivo OMA 1.2. Esta plataforma la incluyen teléfonos como N75, N76, E90 y N95 de Nokia.

Symbian OS 9.3: Lanzado el 12 de Julio de 2006. Las mejoras incluyen un mejor aprovechamiento de la memoria, ayuda nativa para Wifi 802.11, HSDPA e incluye el vietnamita entre sus posibles opciones de lengua.

Symbian OS 9.5: Anunciado en Marzo de 2007. Sus características redujeron el uso de la memoria RAM hasta en un 25% dando como resultado una mayor duración de la batería. Las aplicaciones son lanzadas hasta un 75% más rápido. Incluye una serie de mejoras para el visionado de televisión digital en formatos DVB-H e ISDB-T y también servicios de localización (GPS). Además incluye un modulo para dar soporte a SQL proveniente de SQLite.

Figura 31. Logotipo de la versión 9.1 del SO Symbian

2.4 Plataformas Subyacentes

Existen dos grandes plataformas distintas en Symbian y que ya han sido comentadas por encima en secciones anteriores. La primera de ellas es Series “xx” desarrollada por Nokia y basada en tecnología genérica (GT) de Symbian. La segunda es UIQ, una plataforma en principio independiente, aunque en sus inicios una de las compañías que más participo en su creación fue Ericsson, al igual que Series “xx” esta basada en tecnología genérica (GT) de Symbian. Estas plataformas difieren en la manera de mostrar las aplicaciones básicamente, son interfaces adecuadas a cada modelo de terminal.

A continuación se muestra una figura con la Arquitectura Symbian y una breve explicación de cada parte de la misma:

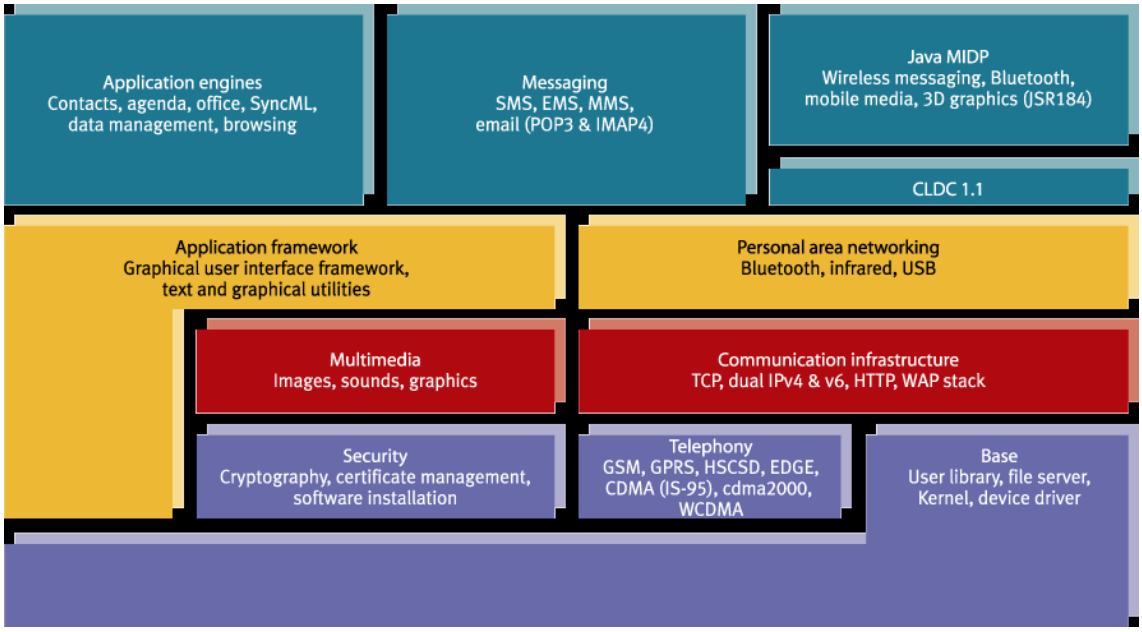


Figura 32. Arquitectura Symbian

La figura ilustra la dependencia entre los componentes del sistema:

- Los componentes de las capas superiores dependen de los componentes de las capas inferiores.

- La *Base* incluye los componentes básicos de todo sistema operativo: kernel, gestión de la memoria y de los procesos, sistema de ficheros, manejadores de dispositivos, seguridad de bajo nivel, librerías básicas de usuario, etc.
- El Gestor de Telefonía realiza la gestión de los sistemas móviles celulares.

- Las pilas de infraestructura de comunicaciones y de red incluyen TCP/IP, GSM, GPRS, WAP, infrarrojos, Bluetooth y comunicaciones serie.

- El Gestor Multimedia se encarga tanto de gestionar la reproducción y grabación de audio como las diferentes funcionalidades de la imagen.
- El Gestor de la Seguridad se encarga de gestionar la seguridad relacionada con la descarga e instalación de las aplicaciones.

- El framework para aplicaciones contiene las librerías para la gestión de datos, texto, portapapeles, gráficos, internacionalización, MMI.
- El Gestor de Mensajería realiza la gestión del correo electrónico, los mensajes de texto, las aplicaciones de serie (menús, agenda, contactos, etc.)

2.4.1 Versiones Series "XX"

Series 40: Esta nueva plataforma creada por Nokia esta diseñada para terminales masivos, esta siendo incorporada a un gran numero de teléfonos pensados para un gran espectro de personas. Incorpora MIDP 2.1 y todas las funcionalidades de Java, además de nuevos conceptos en cuanto a diseño de su interfaz. Es utilizada también en terminales de gama alta y exclusiva por su bajo consumo de recursos.

Series 60: Esta destinada a usarse con teclado numérico exclusivamente. Se caracteriza por una menor resolución de pantalla (176x208), aunque actualmente con la tercera edición de esta interfaz ya no tiene una resolución única y depende en gran medida del terminal que incorpore esta tecnología. Es la más extendida de todas las plataformas que funcionan bajo Symbian, y se compone a su vez de tres versiones (first, second and third edition). Nokia es el creador de Series 60 pero lo ha licenciado a otras compañías como LG, que pueden customizar la pantalla, sonidos, animaciones, etc.

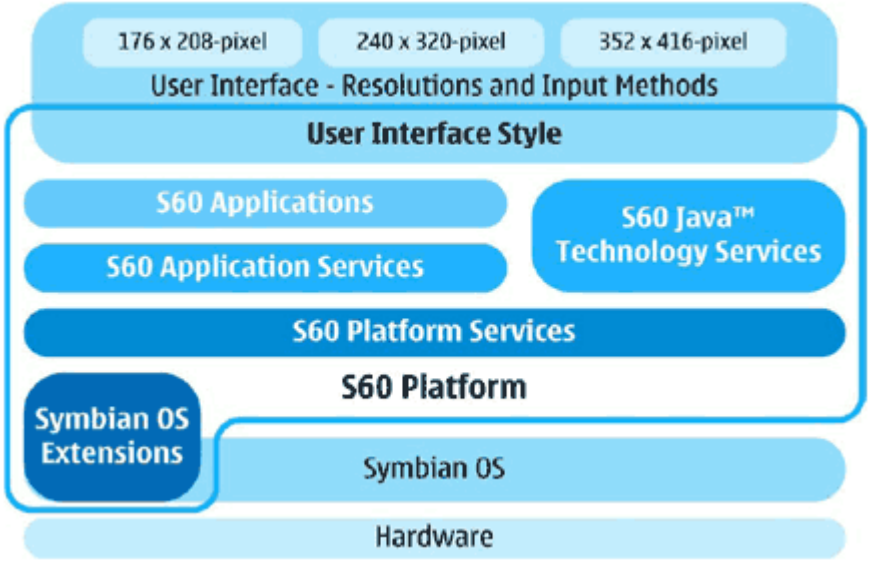


Figura 33. Plataforma Series 60 de Nokia

Series 80:Diseñada para teclados de tipo qwerty y para una utilización con las dos manos. Esta especialmente diseñada para uso profesional, los terminales que la incorporan utilizan funcionalides habitualmente incluidas en los PDA´s. En un principio era utilizado sobre Symbian 6.0, actualmente se rehízo para poder ser usada sobre Symbian 7.0s.

Series 90: Utilizada sobre terminales con una pantalla grande y táctil (similar a UIQ), creada para funcionar sobre Symbian 7.0s.

2.4.2 Versiones UIQ

UIQ es un interfaz, una plataforma flexible y customizable de desarrollo. Esta integrado y probado con Symbian OS, el sistema operativo estándar de la industria principal de los SmartPhones. En sus inicios estaba destinada para utilizarse en terminales con pantallas táctiles, con altas resoluciones y teclado numérico opcional.

A continuación se muestra una figura con la integración de la tecnología UIQ dentro del Sistema Operativo Symbian:

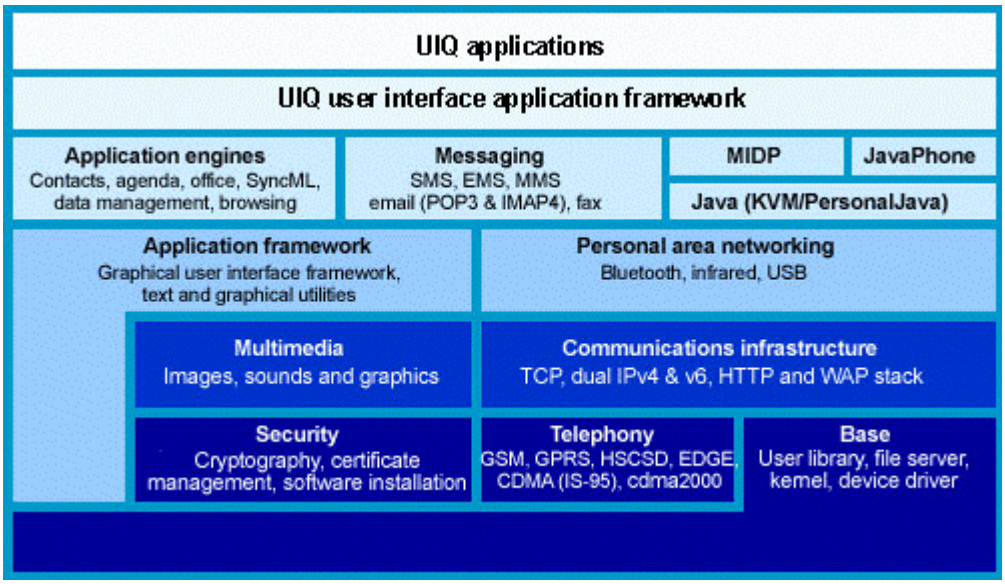


Figura 34. Plataforma UIQ

UIQ 1.0: (Lanzamiento en el año 2000)

- Nuevo interfaz VGA (pixeles 240*320), para OS v6.0 de Symbian.
- Capacidades integradas de la telefonía.
- PersonalJava 1.0 y Javaphone 1.0.

UIQ 1.1: (Lanzamiento en el año 2001)

- Para OS v6.1 de Symbian.
- Triple-Banda GSM, HSCSD, y GPRS.
- Bluetooth v1.0b.
- WAP 1.2.1.

UIQ 2.0: (Lanzamiento en el año 2002)

- Para OS v7.0 de Symbian.
- Telefonía con varios modos de funcionamiento API con CDMA.
- Tamaño de pantalla flexible - permite configuraciones más pequeñas.
- PersonalJava 1.1 y Java MIDP 1.0.
- Mejoras/adiciones de estándares móviles, tales como MMS, Bluetooth, SyncML, Web.

UIQ 2.1: (Lanzamiento en el año 2003)

- Para OS v7.0 de Symbian.
- Java MIDP 2.0.
- Uso integrado de la telefonía.
- Mejora de SyncML.
- Gerencia de SIM.
- Temas.

Figura 35. Logo Corporativo de la plataforma UIQ

UIQ 3: (Lanzamiento en el año 2005)

- Para OS v9 .1 de Symbian.
- Incluye configuraciones diferentes para el interfaz del usuario..
- Uso de una interfaz de usuario enriquecida.

UIQ 3.1: (Lanzamiento en el año 2007)

- Para OS v9.2 de Symbian.
- SVG minúsculo, gráfico del vector escalable, para una experiencia gráfica más rica.
- Una flexibilidad más amplia en las configuraciones del softkey.
- Lista de tareas que facilitan la navegación del usuario.

Figura 36. Diagrama con las funcionalidades de la versión 3.2 de UIQ

2.4.3 Similitudes y diferencias entre Series 60 y UIQ

Existen dos interfaces de usuario diferenciadas en el consorcio Symbian: la plataforma Series 60 impulsada por Nokia y la plataforma UIQ desarrollada por Sony Ericsson. Ambas están disponibles para que otros fabricantes puedan obtener una licencia y basar en ellas el diseño de sus propios terminales Symbian. La principal diferencia entre ambas es que la Series 60 está pensada para que el usuario interactúe con el terminal mediante un teclado, mientras que la UIQ está diseñada para ser manejada mediante un lápiz y una pantalla táctil. Ambas plataformas incluyen, además de la especificación de los elementos propios de la interfaz de usuario, un conjunto de aplicaciones estándar que realizan las tareas más comunes en un terminal móvil multimedia.

El núcleo de la interfaz de usuario es suministrado por Symbian y se llama Uikon. Este incluye eikcore.dll (proporciona las clases principales para el marco de trabajo) y eikcoctl.dll (proporciona las clases encargadas del manejo de botones, menús y listas desplegables). Pero. A parte de esto, cada plataforma tiene sus propias librerías especializadas en la interfaz de usuario, construidas sobre los módulos suministrados por Uikon. Mientras que para la interfaz de usuario en UIQ la librería se llama Qikon, su correspondiente librería en Series 60 es Avkon.

Otro aspecto a reseñar es que estas interfaces requieren versiones diferentes del SO Symbian para su ejecución. En este sentido, la plataforma Series 60 funciona sobre la versión 6.1 (concretamente, la versión 1.0. de la Series 60) y la versión 7.0s (concretamente, la versión 2.0 de la Series 60) del SO, y la plataforma UIQ funciona sobre la versión 7.0. del SO.

La interfaz gráfica Series 60

La interfaz Series 60 requiere una pantalla en color de tamaño predeterminado (176 x 208 píxeles), e incluye soporte para realizar una serie de funciones especiales, por lo cual dispone de dos teclas configurables por software (*softkeys*), de *joystick* de cuatro direcciones, de lanzamiento de aplicaciones, de rotación de aplicaciones (para pasar de una aplicación en ejecución a otra) y de teclas de inicio y finalización de llamada. También incluye teclas de borrado y edición.

El teclado es de tipo numérico con doce teclas.

En esta interfaz se recomienda que el procesador sea ARM de 32 bit y que el tamaño de las memorias ROM y RAM sea de 16 Mbyte y 8 Mbyte, respectivamente.

Otros fabricantes de terminales, que no sean Nokia y Sony, también tienen la posibilidad de personalizar la apariencia de la interfaz Series 60 en aspectos como:

- Poder reemplazar todos los *bitmaps* ofrecidos por Nokia por los gráficos propios del fabricante.
- Poder integrar sonidos, animaciones, esquemas de colores y fuentes específicos del fabricante
- Incluir nuevas teclas e indicadores.
- Incluir nuevos servicios y aplicaciones.

Además, el usuario puede personalizar la apariencia de la interfaz mediante el empleo de temas que modifiquen determinados aspectos del terminal, como los iconos, los fondos de pantalla, los esquemas de colores, etc. Del mismo modo, también es posible modificar las funciones asociadas a las teclas, definir accesos directos (o secuencias) asociados a teclas específicas o modificar los menús. Estas capacidades ofrecen un conjunto de oportunidades de personalización de la interfaz por parte del operador.

Las aplicaciones de utilidad general de la plataforma Series 60 son las correspondientes a agenda, calendario, tareas, bloc de notas, álbum de fotos, cámara, reproductoras de audio y vídeo, compositor musical, reloj, calculadora, conversor de unidades, grabador de voz, juegos, etc. También incluye otras aplicaciones, como son:

- La gestión del teléfono (configuración, registros de llamadas, perfiles de usuario, marcación rápida, marcación por voz, tonos asociados a llamada, etc.).
- La descarga, gestión e instalación de las aplicaciones. Las descargas se pueden realizar a través de múltiples interfaces: Bluetooth, WAP, correo, SMS/MMS, USB, etc.
- La descarga de contenido multimedia.
- La posibilidad de disponer de cliente de correo electrónico.
- La posibilidad de disponer de navegador WAP y WEB
- La posibilidad de realizar la sincronización remota con el PC.

La diferencia mas reseñable es el aspecto de la pantalla en cada una de las versiones (tanto Series 60 como UIQ). En la siguiente figura se puede observar la estructura de la pantalla de un dispositivo perteneciente a Series 60, en ella se explican las diferentes zonas en que se divide:

Panel de Estado

Panel Principal

Panel de Control

Figura 37. Pantalla de la interfaz series 60

Panel de Estado: Muestra información general como potencia de la señal recibida, batería y el titulo de la sección en la que estamos actualmente.

Panel Principal: Es usado para mostrar información de las aplicaciones y poder acceder a ellas.

Panel de Control: Este panel nos sirve para poder navegar por las opciones de cada aplicación a través del teclado numérico. Con el accederemos a los ajustes de los programas, apertura y cierre de los mismo, etc.

La interfaz gráfica UIQ

UIQ es una interfaz gráfica personalizable basada en pantalla táctil y lápiz, y que se ejecuta sobre el SO Symbian. Los fabricantes no incluidos en el consorcio Symbian pueden obtener una licencia para desarrollar sus propios terminales a través de la plataforma UIQ.

Una versión muy extendida de UIQ es la 2.1., que permite distintas configuraciones. La pantalla tiene dos opciones diferentes de tamaño: el tipo *Communicator* de 240 x 320 píxeles (1/4 VGA) o el tipo *Smartphone* de 208 x 320 píxeles, aunque también es posible definir nuevas configuraciones en el rango comprendido entre 208 y 240 píxeles.

Esta versión de UIQ dispone de un módulo de telefonía integrado y puede emplear una configuración PDA para que cualquier dispositivo pueda ser desarrollado sin el módulo de telefonía.

Tanto el usuario como las empresas que disponen de licencia pueden personalizar la interfaz UIQ mediante el uso de un menú de temas que incluyen un fondo de pantalla, esquema de colores y sonidos. El fabricante con licencia puede también personalizar determinados elementos, como el selector y el lanzador de aplicaciones, la pantalla inicial y los ficheros de ayuda.

El hardware de referencia para la implementación de UIQ es un procesador Intel StrongArm, que disponga de 64 Mbyte de memoria RAM y 32 Mbyte de memoria Flash (ROM), IrDA, USB, bahías PCMCIA y Compact Flash, y dos puertos de comunicaciones RS232. El tamaño de punto de la pantalla debe estar comprendido entre 0,192 y 0,24 pulgadas, y el color puede variar entre 8 bit y 24 bit de resolución. También son necesarias las teclas que realizan la función de Arriba, Abajo y Confirmar, y son opcionales las que realizan la función de Derecha, Izquierda y Grabar, además de otras teclas que quiera incluir el poseedor de la licencia.

UIQ incluye un conjunto de aplicaciones y funciones similares a los terminales Series 60. El usuario las elige mediante el selector (situado en la parte superior de la pantalla) o el lanzador de aplicaciones.

En la siguiente figura se observa la estructura de pantalla de un terminal Sony Ericsson (P990), esta es de UIQ versión 3, es un poco diferente de la versión UIQ del P800 pero en esencia se divide en las mismas subzonas, las cuales se explicarán a continuación:

Figura 38. Pantalla de la interfaz UIQ versión 3

Selector de Aplicaciones

En este caso, esta figura es de UIQ versión 3, en las versiones anteriores aquí se mostraban un conjunto de iconos para un acceso rápido a las aplicaciones más comunes. Como podemos comprobar en esta versión este se ha sustituido por un panel desplegable, dentro del mismo encontraremos esos iconos pero en un entorno mas compacto y amigable.

Barra de Menú

Contiene las diferentes opciones de la aplicación.

Espacio de Aplicación

Las aplicaciones utilizan esta área central de la pantalla para mostrar información.

Barra de Estado

En ella, como en el caso del Panel de Estado de las Series 60, se muestra información general como potencia de la señal recibida, activación de sistemas como bluetooth, la llegada de nuevo sms, cantidad de batería restante, etc.

2.5 Symbian OS y Sony Ericsson P800

El Sony Ericsson P800 basado en el sistema operativo Symbian dispone de 3 tipos de memoria: RAM, Flash y externa (Memory Stick).

- La memoria RAM (Random Access) es controlada por el sistema operativo Symbian y no se puede usar ni para almacenar datos ni aplicaciones.

- De memoria Flash encontramos 2 bancos:

    1. El primer banco es usado como ROM (Z:). Contiene el sistema operativo Symbian, incorpora aplicaciones e información multimedia. También incorpora el lenguaje que por defecto va instalado en el P800, el ingles británico.

    2. El segundo banco esta dividido en dos partes. La primera de ellas es la unidad C: y la otra parte esta reservada (extensión de la ROM).

- Adicionalmente el P800 da la posibilidad de aumentar la capacidad de la memoria con la incorporación de una memoria externa de hasta 128 MB.

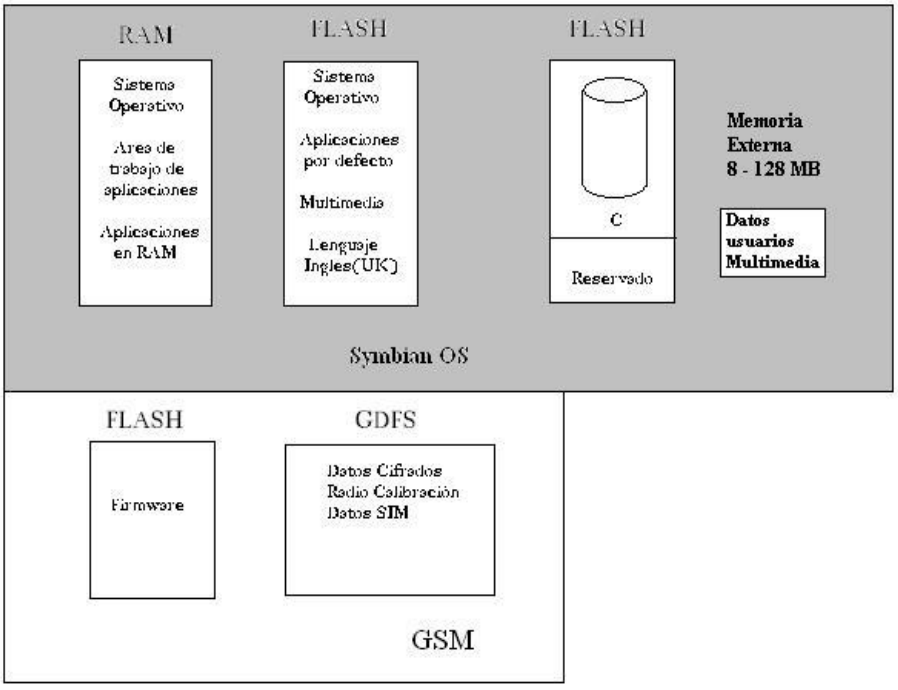


Figura 39. División de memoria en Sony Ericsson P800

2.5.1 Unidad C:

La unidad C: del Sony Ericsson P800 se divide en 3 zonas:

1. La primera tiene una capacidad de 5,7 Mb, que es la destinada a almacenar las aplicaciones instaladas y los datos de usuario.

2. La segunda es la destinada a almacenar contenido multimedia. Esta tiene una capacidad de 3 MB.

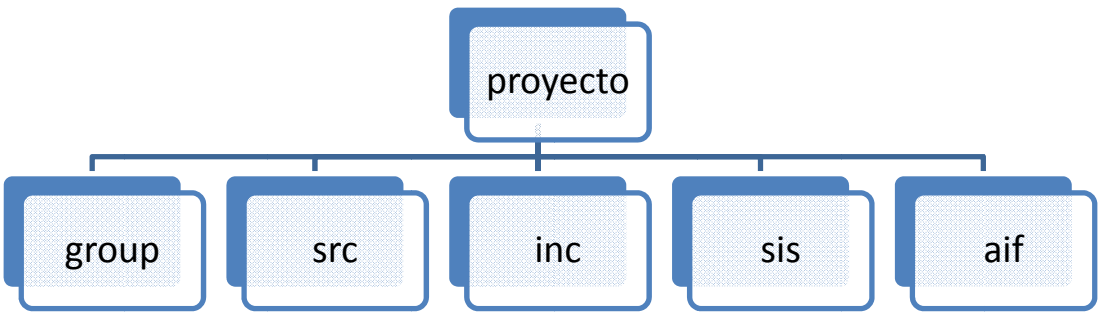
3. La última parte, de una capacidad de 2,6 Mb, es en la que se almacenan los distintos idiomas en los que se puede configurar el P800. Todos los idiomas son borrables, excepto el inglés británico que esta instalado por defecto. Cada idioma instalado tiene un tamaño de alrededor de 650 Kb.

A través de la conexión del Sony Ericsson P800 al ordenador podemos acceder a la unidad C: del móvil. Esta nos aparecerá nombrada como "Memoria del teléfono" y dividida en varias carpetas. Se podrá acceder al contenido de esas carpetas añadiendo o eliminando archivos a gusto del usuario. También se podrán crear subcarpetas dentro de cada una de las carpetas que por defecto aparecen.

2.6 Estructura del directorio

Según el lenguaje de programación que utilicemos, Symbian dispone de una estructura para todas las aplicaciones. Estas se dividen en subdirectorios pertenecientes a un directorio global (project) en las cuales se distribuyen los diferentes archivos (cabeceras, códigos fuente, librerías, etc.). Ahora se verán las dos estructuras básicas con las que se trabajará en ese proyecto.

La estructura de un proyecto Symbian utilizando C++ será la siguiente:



Directorio group: De este directorio cuelgan los archivos referentes a la compilación e importación de proyectos (.inf, .mmp, .rss, .resources).

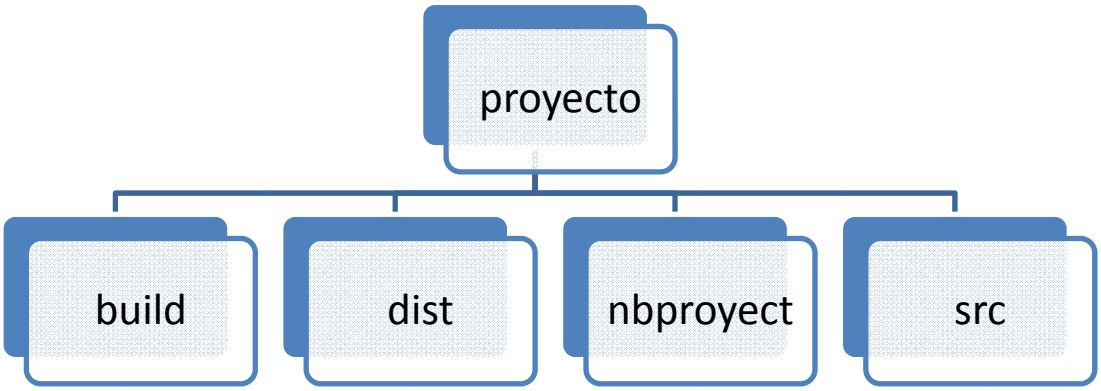
Directorio src: Contiene los archivos fuente del programa (.cpp).

Directorio inc: Contiene las cabeceras de los archivos fuente que se encuentran en src (.hrh, .h).

Directorio sis: Contiene el archivo .pkg necesario para la creación del archivo de instalación en el móvil. Este archivo es muy importante ya que proporciona información como la ubicación que tendrá la aplicación una vez instalada en el terminal o el tipo de plataforma para que esta aplicación haya sido programada. Una vez creado el archivo .sis también se ubicará en esta carpeta.

Directorio aif: Este subdirectorio es opcional, no aparece en todos los proyectos, a veces su contenido se ubica en otro subdirectorio llamado data. Contiene el archivo .rss, el cual proporciona parámetros como los distintos lenguajes en los que se puede mostrar la aplicación, identificador de la aplicación, posibilidad de adjuntar algún icono acompañando al nombre de la aplicación, etc.

Para el caso de un proyecto en Java, su árbol de directorios será el siguiente (utilizando Netbeans como IDE):



Directorio build: Este subdirectorio es similar al group del caso anterior. En el estarán los archivos referentes a la compilación (.class) e importación del proyecto. Además de algunas subdirectorios que se generan dentro del mismo al realizar la compilación (estos sirven para utilizar el debugador, emulador, etc.).

Directorio dist: Este es similar al subdirectorio sis. En el se guardan los archivos que vamos a utilizar para la instalación de la aplicación en el móvil (.jad para probar en el emulador, .jar el instalador para el terminal).

Directorio nbproyect (Significa Netbeans proyect): se almacenan archivos necesarios para la apertura y manejo del proyecto dentro del IDE Netbeans, son del tipo .xml y .properties. Contienen propiedades y parámetros necesarios para la correcta manipulación del proyecto.

Directorio src: Como su propio nombre indica es similar al src del caso de C++. Contiene los archivos fuente de la aplicación (.java) y en el caso de que el programa necesitara de imágenes, dentro de este subdirectorio existirá otro subdirectorio llamado icons que contendrá todas las imágenes.

Capitulo 3

Java

3.1 Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 1990. Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

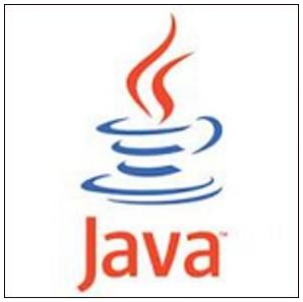


Figura 40. Logotipo de Java (Sun)

El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel como punteros. JavaScript, un lenguaje interpretado, comparte un nombre similar y una sintaxis similar, pero no está directamente relacionado con Java.

Sun Microsystems proporciona una implementación GNU General Public License de un compilador Java y una máquina virtual Java, conforme a las especificaciones del Java Community Process, aunque la biblioteca de clases que se requiere para ejecutar los programas Java no es software libre.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre.

3.1.1 Historia del lenguaje Java

La tecnología Java se creó como una herramienta de programación en una pequeña operación secreta y anónima denominada "the Green Project" en Sun Microsystems en el año 1991.

El equipo secreto ("Green Team"), compuesto por trece personas y dirigido por James Gosling, se encerró en una oficina desconocida de Sand Hill Road en Menlo Park, interrumpió todas las comunicaciones regulares con Sun y trabajó sin descanso durante 18 meses.

Intentaban anticiparse y prepararse para el futuro de la informática. Su conclusión inicial fue que al menos en parte se tendería hacia la convergencia de los dispositivos digitales y los ordenadores.

Figura 41. Duke, la mascota de Java.

Tres de las principales razones que llevaron a crear Java son:

1. Creciente necesidad de interfaces mucho más cómodas e intuitivas que los sistemas de ventanas que proliferaban hasta el momento.
2. Fiabilidad del código y facilidad de desarrollo. Gosling observó que muchas de las características que ofrecían C o C++ aumentaban de forma alarmante el gran coste de pruebas y depuración. Por ello en los sus ratos libres creó un lenguaje de programación donde intentaba solucionar los fallos que encontraba en C++.
3. Enorme diversidad de controladores electrónicos. Los dispositivos electrónicos se controlan mediante la utilización de microprocesadores de bajo precio y reducidas prestaciones, que varían cada poco tiempo y que utilizan diversos conjuntos de instrucciones. Java permite escribir un código común para todos los dispositivos.

El resultado fue un lenguaje de programación que no dependía de los dispositivos denominado "Oak".

Para demostrar cómo podía contribuir este nuevo lenguaje al futuro de los dispositivos digitales, el equipo desarrolló un controlador de dispositivos de mano para uso doméstico destinado al sector de la televisión digital por cable. Por desgracia, la idea resultó ser demasiado avanzada para el momento y el sector de la televisión digital por cable no estaba listo para el gran avance que la tecnología Java les ofrecía.

Después de unos años sin trascendencia, en los que Java estuvo un poco sumido en el olvido Bill Joy (cofundador de Sun y uno de los desarrolladores principales del sistema operativo Unix de Berckley) creyó que era el momento de que asumiera un papel tecnológico principal. Joy juzgó que Internet podría llegar a ser el campo adecuado para disputar a Microsoft su primacía en el terreno del software, y vio en Oak el instrumento idóneo para llevar a cabo estos planes.

Para poder presentarlo en sociedad se tuvo que modificar el nombre de este lenguaje de programación y se tuvo que realizar una serie de modificaciones de diseño para poderlo adaptar al propósito mencionado. Así Java fue presentado en sociedad en agosto de 1995.

Algunas de las razones que llevaron a Bill Joy a pensar que Java podría llegar a ser rentable son:

- Java es un lenguaje orientado a objetos: Esto es lo que facilita abordar la resolución de cualquier tipo de problema.
- Es un lenguaje sencillo, aunque sin duda potente.
- La ejecución del código Java es segura y fiable: Los programas no acceden directamente a la memoria del ordenador, siendo imposible que un programa escrito en Java pueda acceder a los recursos del ordenador sin que esta operación le sea permitida de forma explícita. De este modo, los datos del usuario quedan a salvo de la existencia de virus escritos en Java. La ejecución segura y controlada del código Java es una característica única, que no puede encontrarse en ninguna otra tecnología.
- Es totalmente multiplataforma: Es un lenguaje sencillo, por lo que el entorno necesario para su ejecución es de pequeño tamaño y puede adaptarse incluso al interior de un navegador.

Poco tiempo después Internet estaba listo para la tecnología Java y, justo a tiempo para su presentación en público en 1995, el equipo pudo anunciar que el navegador Netscape Navigator incorporaría la tecnología Java.

Actualmente, a punto de cumplir los 10 años de existencia, la plataforma Java ha atraído a cerca de 4 millones de desarrolladores de software, se utiliza en los principales sectores de la industria de todo el mundo y está presente en un gran número de dispositivos, ordenadores y redes de cualquier tecnología de programación.

De hecho, su versatilidad y eficiencia, la portabilidad de su plataforma y la seguridad que aporta, la han convertido en la tecnología ideal para su aplicación a redes, de manera que hoy en día, más de 2.500 millones de dispositivos utilizan la tecnología Java. Más de 700 millones de ordenadores 708 millones de teléfonos móviles y otros dispositivos de mano (fuente: Ovum) 1000 millones de tarjetas inteligentes Además de sintonizadores, impresoras, web cams, juegos, sistemas de navegación para automóviles, terminales de lotería, dispositivos médicos, cajeros de pago en aparcamientos, etc. Hoy en día, puede encontrar la tecnología Java en redes y dispositivos que comprenden desde Internet y superordenadores científicos hasta portátiles y teléfonos móviles; desde simuladores de mercado en Wall Street hasta juegos de uso doméstico y tarjetas de crédito: Java está en todas partes.

3.1.2 Futuro de Java

Existen muchas críticas a Java debido a su lenta velocidad de ejecución, aproximadamente unas 20 veces más lento que un programa en lenguaje C. Sun está trabajando intensamente en crear versiones de Java con una velocidad mayor.

El problema fundamental de Java es que utiliza una representación intermedia denominada *código de byte* para solventar los problemas de portabilidad. Los *códigos de byte* posteriormente se tendrán que transformar en código máquina en cada máquina en que son utilizados, lo que ralentiza considerablemente el proceso de ejecución.

La solución que se deriva de esto parece bastante obvia: fabricar ordenadores capaces de comprender directamente los códigos de byte. Éstas serían unas máquinas que utilizaran Java como sistema operativo y que no requerirían en principio de disco duro porque obtendrían sus recursos de la red.

A los ordenadores que utilizan Java como sistema operativo se les llama Network Computer, WebPC o WebTop. La primera gran empresa que ha apostado por este tipo de máquinas ha sido Oracle, que en enero de 1996 presentó en Japón su primer NC (Network Computer), basado en un procesador RISC con 8 Megabytes de RAM. Tras Oracle, han sido compañías del tamaño de Sun, Apple e IBM las que han anunciado desarrollos similares.

La principal empresa en el mundo del software, Microsoft, que en los comienzos de Java no estaba a favor de su utilización, ha licenciado Java, lo ha incluido en Internet Explorer (versión 3.0 y posteriores), y ha lanzado un entorno de desarrollo para Java, que se denomina Visual J++.

El único problema aparente es la seguridad para que Java se pueda utilizar para transacciones críticas. Sun va a apostar por firmas digitales, que serán clave en el desarrollo no sólo de Java, sino de Internet.

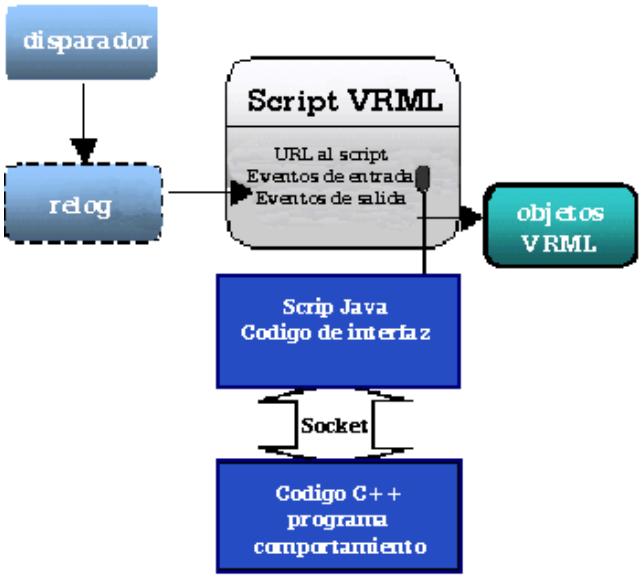


Figura 42. Diagrama de ejemplo de una aplicación Java

3.1.3 Java e Internet

Entre junio y julio de 1994, tras una sesión maratónica de tres días entre John Gaga, James Gosling, Joy Naughton, Wayne Rosing y Eric Schmidt, el equipo reorientó la plataforma hacia la Web. Sintieron que la llegada del navegador Web Mosaic, propiciaría que Internet se convirtiese en un medio interactivo, como el que pensaban era la televisión por cable. Naughton creó entonces un prototipo de navegador, WebRunner, que más tarde sería conocido como HotJava.

Ese año renombraron el lenguaje como **Java** tras descubrir que "Oak" era ya una marca comercial registrada para adaptadores de tarjetas gráficas. El término Java fue acuñado en una cafetería frecuentada por algunos de los miembros del equipo. Pero no está claro si es un acrónimo o no, aunque algunas fuentes señalan que podría tratarse de las iniciales de sus creadores: ***J**ames Gosling, **A**rthur **V**an Hoff, y **A**ndy Bechtolsheim*. Otros abogan por el siguiente acrónimo, ***J**ust **A**nother **V**ague **A**cronym* ("sólo otro acrónimo ambiguo más"). La hipótesis que más fuerza tiene es la que Java debe su nombre a un tipo de café disponible en la cafetería cercana. Un pequeño signo que da fuerza a esta teoría es que los 4 primeros bytes (el *número mágico*) de los archivos .class que genera el compilador, son en hexadecimal, 0xCAFEBABE.

En octubre de 1994, se les hizo una demostración de HotJava y la plataforma Java a los ejecutivos de Sun. Java 1.0a pudo descargarse por primera vez en 1994, pero hubo que esperar al 23 de mayo de 1995, durante las conferencias de SunWorld, a que vieran la luz pública Java y HotJava, el navegador Web. El acontecimiento fue anunciado por John Gage, el Director Científico de Sun Microsystems. El acto estuvo acompañado por una pequeña sorpresa adicional, el anuncio por parte de Marc Andreessen, Vicepresidente Ejecutivo de Netscape, que Java sería soportado en sus navegadores. El 9 de enero del año siguiente, 1996, Sun fundó el grupo empresarial JavaSoft para que se encargase del desarrollo tecnológico. Dos semanas más tarde la primera versión de Java fue publicada.

Java en el cliente

La capacidad de la continuidad del uso de Java por el gran público. Flash está más extendido para animaciones interactivas y los desarrolladores estén empezando a usar la tecnología AJAX también en este campo. Java suele usarse para aplicaciones más complejas como la zona de juegos de Yahoo, Yahoo! Games, o reproductores de video.

Java en el servidor

En la parte del servidor, Java es más popular que nunca, con muchos sitios empleando páginas JavaServer, conectores como Tomcat para Apache y otras tecnologías Java.

Java en el PC de escritorio

Aunque cada vez la tecnología Java se acerca más y más al PC de sobremesa, las aplicaciones Java han sido relativamente raras para uso doméstico, por varias razones.

- Las aplicaciones Java pueden necesitar gran cantidad de memoria física.

- La Interfaz Gráfica de Usuario (GUI) no sigue de forma estricta la *Guía para Interfaces Humana* (Human Interface Guidelines), así como tampoco aquella a la que estamos habitualmente acostumbrados. La apariencia de las fuentes no tiene las opciones de optimización activadas por defecto, lo que hace aparecer al texto como si fuera de baja calidad.
- Las herramientas con que cuenta el JDK no son suficientemente potentes para construir de forma simple aplicaciones potentes. Aunque el uso de herramientas como Eclipse, un IDE con licencia libre de alta calidad, facilita enormemente las tareas de desarrollo.
- Hay varias versiones del Entorno en Tiempo de Ejecución de Java, el JRE. Es necesario tener instalada la versión adecuada.
- Las aplicaciones basadas en la Web están tomando la delantera frente a aquellas que funcionan como entidades independientes. Las nuevas técnicas de programación producen aplicaciones basadas en un modelo en red cada vez más potentes.

Sin embargo hay aplicaciones Java cuyo uso está ampliamente extendido, como NetBeans, el entorno de desarrollo (IDE) Eclipse, y otros programas como LimeWire y Azureus para intercambio de archivos. Java también es el motor que usa MATLAB para el renderizado de la interfaz gráfica y para parte del motor de cálculo. Las aplicaciones de escritorio basadas en la tecnología Swing y SWT (Standard Widget Toolkit) suponen una alternativa a la plataforma .Net de Microsoft.

Disponibilidad del JRE de Java

Una versión del JRE (Java Runtime Environment) está disponible en la mayoría de equipos de escritorio. Sin embargo, Microsoft no lo ha incluido por defecto en sus sistemas operativos. En el caso de Apple, éste incluye una versión propia del JRE en su sistema operativo, el Mac OS. También es un producto que por defecto aparece en la mayoría de las distribuciones de Linux. Debido a incompatibilidades entre distintas versiones del JRE, muchas aplicaciones prefieren instalar su propia copia del JRE antes que confiar su suerte a la aplicación instalada por defecto. Los desarrolladores de applets de Java o bien deben insistir a los usuarios en la actualización del JRE, o bien desarrollar bajo una versión antigua de Java y verificar el correcto funcionamiento en las versiones posteriores.

3.1.4 Historial de versiones

Java ha experimentado numerosos cambios desde la versión primigenia, JDK 1.0, así como un enorme incremento en el número de clases y paquetes que componen la librería estándar. Desde J2SE 1.4, la evolución del lenguaje ha sido regulada por el JCP (Java Community Process), que usa *Java Specification Requests* (JSRs) para proponer y especificar cambios en la plataforma Java. El lenguaje en sí mismo está especificado en la *Java Language Specification* (JLS), o Especificación del Lenguaje Java. Los cambios en los JLS son gestionados en JSR 901.

- **JDK 1.0** (23 de enero de 1996) — Primer lanzamiento.

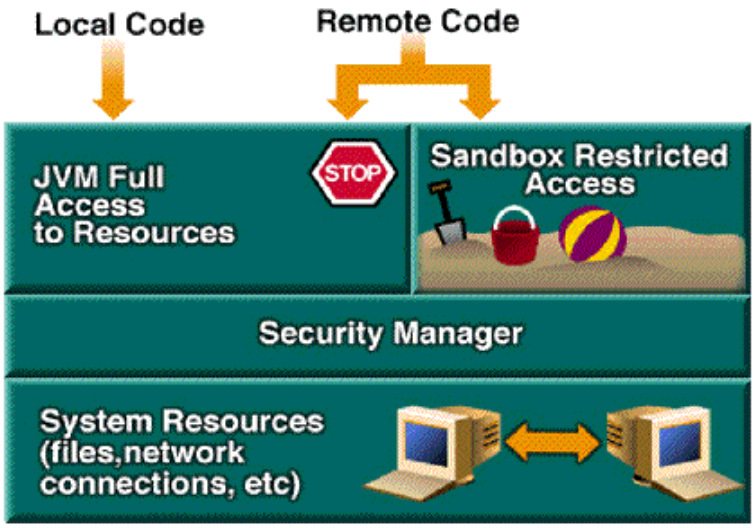


Figura 43. Esquema de Seguridad en JDK 1.0

- **JDK 1.1** (19 de febrero de 1997) — Principales adiciones incluidas:
    - Una reestructuración intensiva del modelo de eventos AWT (Abstract Windowing Toolkit)
    - Clases internas (inner classes)
    - JavaBeans
    - JDBC (Java Database Connectivity), para la integración de bases de datos
    - RMI (Remote Method Invocation)

- **J2SE 1.2** (8 de diciembre de 1998) — Nombre clave *Playground*. Esta y las siguientes versiones fueron recogidas bajo la denominación Java 2 y el nombre "J2SE" (Java 2 Platform, Standard Edition), reemplazó a JDK para distinguir la plataforma base de J2EE (Java 2 Platform, Enterprise Edition) y J2ME (Java 2 Platform, Micro Edition). Otras mejoras añadidas incluían:
    - La palabra reservada (keyword) strictfp
    - Reflexión en la programación
    - La API gráfica (Swing) fue integrada en las clases básicas
    - La máquina virtual (JVM) de Sun fue equipada con un compilador JIT (Just in Time) por primera vez
    - Java Plug-in
    - Java IDL, una implementación de IDL (Interfaz para Descripción de Lenguaje) para la interoperabilidad con CORBA
    - Colecciones (Collections)

- **J2SE 1.3** (8 de mayo de 2000) — Nombre clave *Kestrel*. Añadia las siguientes novedades:

    - La inclusión de la máquina virtual de HotSpot JVM (la JVM de HotSpot fue Lanzada inicialmente en abril de 1999, para la JVM de J2SE 1.2)
    - RMI fue cambiado para que se basara en CORBA
    - JavaSound
    - Se incluyó el Java Naming and Directory Interface (JNDI) en el paquete de librerías principales (anteriormente disponible como una extensión)
    - Java Platform Debugger Architecture (JPDA)

- **J2SE 1.4** (6 de febrero de 2002) — Nombre Clave *Merlín*. Este fue el primer lanzamiento de la plataforma Java desarrollado bajo el Proceso de la Comunidad Java como JSR 59. Los cambios más notables fueron:

    - Palabra reservada assert (Especificado en JSR 41.)
    - Expresiones regulares modeladas al estilo de las expresiones regulares Perl
    - Encadenación de excepciones Permite a una excepción encapsular la excepción de bajo nivel original.
    - Non-blocking NIO (New Input/Output) (Especificado en JSR 51.)
    - Logging API (Specified in JSR 47.)
    - API I/O para la lectura y escritura de imágenes en formatos como JPEG o PNG
    - Parser XML integrado y procesador XSLT (JAXP) (Especificado en JSR 5 y JSR 63.)
    - Seguridad integrada y extensiones criptográficas (JCE, JSSE, JAAS)
    - Java Web Start incluido (El primer lanzamiento ocurrió en Marzo de 2001 para J2SE 1.3) (Especificado en JSR 56.).

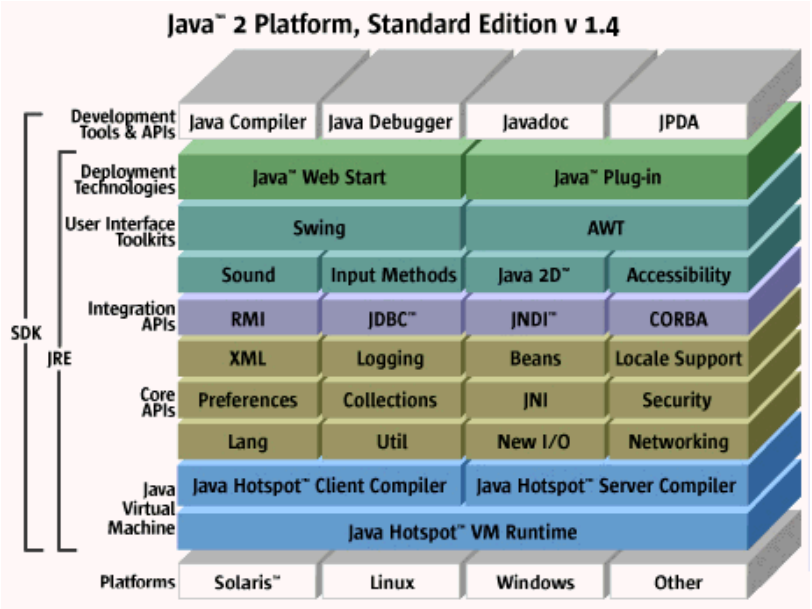


Figura 44. Plataforma J2SE versión 1.4

- **J2SE 5.0** (30 de septiembre de 2004) — Nombre clave: *Tiger*. Desarrollado bajo JSR 176, Tiger añadió un número significativo de nuevas características:

    - Plantillas (genéricos) — provee conversión de tipos (type safety) en tiempo de compilación para colecciones y elimina la necesidad de la mayoría de conversión de tipos (type casting). (Especificado por JSR 14.)
    - Metadatos — también llamados anotaciones, permite a estructuras del lenguaje como las clases o los métodos, ser etiquetados con datos adicionales, que puedan ser procesados posteriormente por utilidades de proceso de metadatos. (Especificado por JSR 175.)
    - Autoboxing/unboxing — Conversiones automáticas entre tipos primitivos (Como los int) y clases de envoltura primitivas (Como Integer). (Especificado por JSR 201.)
    - Enumeraciones — la palabra reservada enum crea una typesafe, lista ordenada de valores (como Dia.LUNES, Dia.MARTES, etc.). Anteriormente, esto solo podía ser llevado a cabo por constantes enteras o clases construidas manualmente (enum pattern). (Especificado por JSR 201.)
    - Varargs (número de argumentos variable) — El último parámetro de un método puede ser declarado con el nombre del tipo seguido por tres puntos. En la llamada al método, puede usarse cualquier número de parámetros de ese tipo, que serán almacenados en un array para pasarlos al método.
    - Bucle for mejorado — La sintaxis para el bucle for se ha extendido con una sintaxis especial para iterar sobre cada miembro de un array o sobre cualquier clase que implemente Iterable, como la clase estándar Collection, de la siguiente forma:

- **Java SE 6** (11 de diciembre de 2006) — Nombre clave *Mustang*. Estuvo en desarrollo bajo la JSR 270. En esta versión, Sun cambió el nombre "J2SE" por **Java SE** y eliminó el ".0" del número de versión. Está disponible en http://java.sun.com/javase/6/. Los cambios más importantes introducidos en esta versión son:

    - Incluye un nuevo marco de trabajo y APIs que hacen posible la combinación de Java con lenguajes dinámicos como PHP, Python, Ruby y JavaScript.
    - Incluye el motor Rhino, de Mozilla, una implementación de Javascript en Java.
    - Incluye un cliente completo de Servicios Web y soporta las últimas especificaciones para Servicios Web, como JAX-WS 2.0, JAXB 2.0, STAX y JAXP.
    - Mejoras en la interfaz gráfica y en el rendimiento.

- **Java SE 7** — Nombre clave *Dolphin*. En el año 2006 aún se encontraba en las primeras etapas de planificación. Se espera que su desarrollo dé comienzo en la primavera de 2006, y se estima su lanzamiento para 2008.

    - Soporte para XML dentro del propio lenguaje.
    - Un nuevo concepto de superpaquete.
    - Soporte para closures.
    - Introducción de anotaciones estándar para detectar fallos en el software.

- No oficiales:
  - NIO2
  - Java Module System.
  - Java Kernel.
  - Nueva API para el manejo de Dias y Fechas, la cual reemplazara las antiguas clases Date y Calendar.
  - Posibilidad de operar con clases BigDecimal usando operandos.

Además de los cambios en el lenguaje, con el paso de los años se han efectuado muchos más cambios dramáticos en la librería de clases de Java (*Java class library*) que ha crecido de unos pocos cientos de clases en JDK 1.0 hasta más de tres mil en J2SE 5.0. APIs completamente nuevas, como Swing y Java2D, han sido introducidas y muchos de los métodos y clases originales de JDK 1.0 están obsoletos.

3.1.5 Filosofía

El lenguaje Java se creó con cinco objetivos principales:

1. Debería usar la metodología de la programación orientada a objetos.
2. Debería permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Debería incluir por defecto soporte para trabajo en red.
4. Debería diseñarse para ejecutar código en sistemas remotos de forma segura.
5. Debería ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objetos, como C++.

Para conseguir la ejecución de código remoto y el soporte de red, los programadores de Java a veces recurren a extensiones como CORBA (Common Object Request Broker Architecture), Internet Communications Engine u OSGi respectivamente.

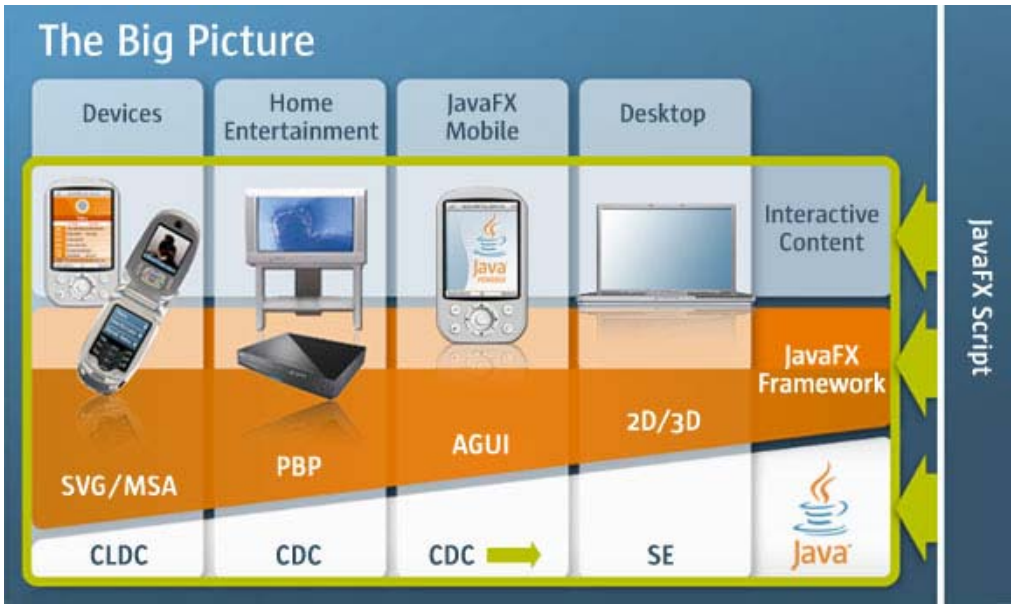


Figura 45. Esquema de las tecnologías Java para cada sistema

Orientado a Objetos

La primera característica, orientado a objetos ("OO"), se refiere a un método de programación y al diseño del lenguaje. Aunque hay muchas interpretaciones para OO, una primera idea es diseñar el software de forma que los distintos tipos de datos que use estén unidos a sus operaciones. Así, los datos y el código (funciones o métodos) se combinan en entidades llamadas objetos. Un objeto puede verse como un paquete que contiene el "comportamiento" (el código) y el "estado" (datos).

El principio es separar aquello que cambia de las cosas que permanecen inalterables. Frecuentemente, cambiar una estructura de datos implica un cambio en el código que opera sobre los mismos, o viceversa. Esta separación en objetos coherentes e independientes ofrece una base más estable para el diseño de un sistema software. El objetivo es hacer que grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de proyectos fallidos. Otra de las grandes promesas de la programación orientada a objetos es la creación de entidades más genéricas (objetos) que permitan la reutilización del software entre proyectos, una de las premisas fundamentales de la Ingeniería del Software.

Un objeto genérico "cliente", por ejemplo, debería en teoría tener el mismo conjunto de comportamiento en diferentes proyectos, sobre todo cuando estos coinciden en cierta medida, algo que suele suceder en las grandes organizaciones. En este sentido, los objetos podrían verse como piezas reutilizables que pueden emplearse en múltiples proyectos distintos, posibilitando así a la industria del software a construir proyectos de envergadura empleando componentes ya existentes y de comprobada calidad; conduciendo esto finalmente a una reducción drástica del tiempo de desarrollo. Podemos usar como ejemplo de objeto el aluminio. Una vez definidos datos (peso, maleabilidad, etc.), y su "comportamiento" (soldar dos piezas, etc.), el objeto "aluminio" puede ser reutilizado en el campo de la construcción, del automóvil, de la aviación, etc.

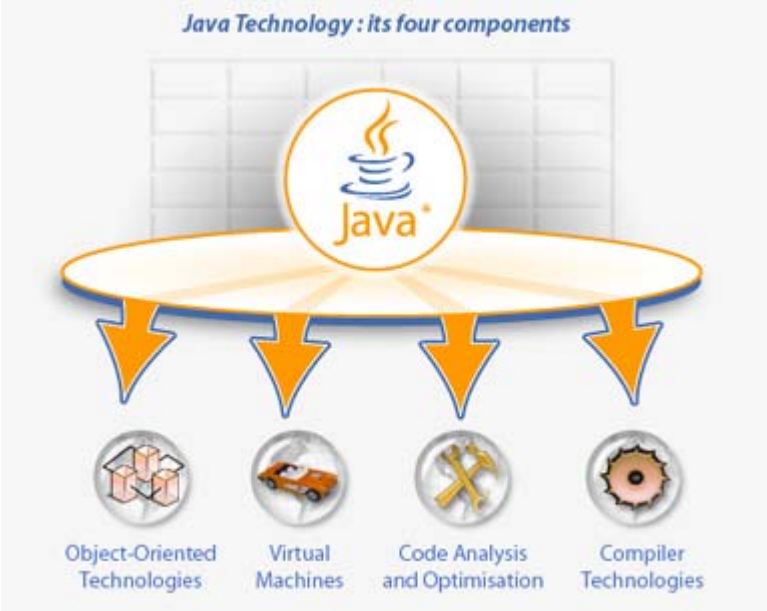


Figura 46. Esquema de la filosofía Java

La reutilización del software ha experimentado resultados dispares, encontrando dos dificultades principales: el diseño de objetos realmente genéricos es pobremente comprendido, y falta una metodología para la amplia comunicación de oportunidades de reutilización. Algunas comunidades de "código abierto" (open source) quieren ayudar en este problema dando medios a los desarrolladores para diseminar la información sobre el uso y versatilidad de objetos reutilizables y librerías de objetos.

Independencia de la plataforma

La segunda característica, la independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware. Es lo que significa ser capaz de escribir un programa una vez y que pueda ejecutarse en cualquier dispositivo, tal como reza el axioma de Java, ‘’’write once, run everywhere’’’.

Para ello, se compila el código fuente escrito en lenguaje Java, para generar un código conocido como "bytecode" (específicamente Java bytecode) instrucciones máquina simplificadas específicas de la plataforma Java. Esta pieza está "a medio camino" entre el código fuente y el código máquina que entiende el dispositivo destino. El bytecode es ejecutado entonces en la máquina virtual (VM), un programa escrito en código nativo de la plataforma destino (que es el que entiende su hardware), que interpreta y ejecuta el código. Además, se suministran librerías adicionales para acceder a las características de cada dispositivo (como los gráficos, ejecución mediante hebras o threads, la interfaz de red) de forma unificada. Se debe tener presente que, aunque hay una etapa explícita de compilación, el bytecode generado es interpretado o convertido a instrucciones máquina del código nativo por el compilador JIT (Just In Time).

Hay implementaciones del compilador de Java que convierten el código fuente directamente en código objeto nativo, como GCJ. Esto elimina la etapa intermedia donde se genera el bytecode, pero la salida de este tipo de compiladores sólo puede ejecutarse en un tipo de arquitectura.

La licencia sobre Java de Sun insiste que todas las implementaciones sean "compatibles". Esto dio lugar a una disputa legal entre Microsoft y Sun, cuando éste último alegó que la implementación de Microsoft no daba soporte a las interfaces RMI y JNI además de haber añadido características ‘’dependientes’’ de su plataforma. Sun demandó a Microsoft y ganó por daños y perjuicios (unos 20 millones de dólares) así como una orden judicial forzando la acatación de la licencia de Sun. Como respuesta, Microsoft no ofrece Java con su versión de sistema operativo, y en recientes versiones de Windows, su navegador Internet Explorer no admite la ejecución de applets sin un conector (o plugin) aparte. Sin embargo, Sun y otras fuentes ofrecen versiones gratuitas para distintas versiones de Windows.

Las primeras implementaciones del lenguaje usaban una máquina virtual interpretada para conseguir la portabilidad. Sin embargo, el resultado eran programas que se ejecutaban comparativamente más lentos que aquellos escritos en C o C++. Esto hizo que Java se ganase una reputación de lento en rendimiento. Las implementaciones

recientes de la JVM dan lugar a programas que se ejecutan considerablemente más rápido que las versiones antiguas, empleando diversas técnicas.

La primera de estas técnicas es simplemente compilar directamente en código nativo como hacen los compiladores tradicionales, eliminando la etapa del bytecode. Esto da lugar a un gran rendimiento en la ejecución, pero tapa el camino a la portabilidad. Otra técnica, conocida como compilación JIT (Just In Time, o ‘’’compilación al vuelo’’’), convierte el bytecode a código nativo cuando se ejecuta la aplicación. Otras máquinas virtuales más sofisticadas usan una ‘’’recompilación dinámica’’’ en la que la VM es capaz de analizar el comportamiento del programa en ejecución y recompila y optimiza las partes críticas. La recompilación dinámica puede lograr mayor grado de optimización que la compilación tradicional (o estática), ya que puede basar su trabajo en el conocimiento que de primera mano tiene sobre el entorno de ejecución y el conjunto de clases cargadas en memoria. La compilación JIT y la recompilación dinámica permiten a los programas Java aprovechar la velocidad de ejecución del código nativo sin por ello perder la ventaja de la portabilidad.

La portabilidad es técnicamente difícil de lograr, y el éxito de Java en ese campo ha sido dispar. Aunque es de hecho posible escribir programas para la plataforma Java que actúen de forma correcta en múltiples plataformas de distinta arquitectura, el gran número de estas con pequeños errores o inconsistencias llevan a que a veces se parodie el eslogan de Sun, "Write once, run anywhere" como "Write once, debug everywhere" (o “Escríbelo una vez, ejecútalo en todas partes” por “Escríbelo una vez, depúralo en todas partes”)

El concepto de independencia de la plataforma de Java cuenta, sin embargo, con un gran éxito en las aplicaciones en el entorno del servidor, como los Servicios Web, los Servlets, los Java Beans, así como en sistemas empotrados basados en OSGi, usando entornos Java empotrados.

Disposición de capas: aplicación nativa

Figura 47. Disposición de capas de una aplicación Java y su independencia de la plataforma

El recolector de basura

Un argumento en contra de lenguajes como C++ es que los programadores se encuentran con la carga añadida de tener que administrar la memoria solicitada dinámicamente de forma manual:

En C++, el desarrollador puede asignar memoria en una zona conocida como *heap* (montículo) para crear cualquier objeto, y posteriormente desalojar el espacio asignado cuando desea borrarlo. Un olvido a la hora de desalojar memoria previamente solicitada puede llevar a una *fuga de memoria*, ya que el sistema operativo seguirá pensando que esa zona de memoria está siendo usada por una aplicación cuando en realidad no es así. Así, un programa mal diseñado podría consumir una cantidad desproporcionada de memoria. Además, si una misma región de memoria es desalojada dos veces el programa puede volverse inestable y llevar a un eventual *cuelgue*. No obstante, se debe señalar que C++ también permite crear objetos en la pila de llamadas de una función o bloque, de forma que se libere la memoria (y se ejecute el destructor del objeto) de forma automática al finalizar la ejecución de la función o bloque.

En Java, este problema potencial es evitado en gran medida por el recolector automático de basura (o *automatic garbage collector*). El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java (Java runtime) es el responsable de gestionar el ciclo de vida de los objetos. El programa, u otros objetos pueden tener localizado un objeto mediante una referencia a éste (que, desde un punto de vista de bajo nivel es una dirección de memoria). Cuando no quedan referencias a un objeto, el recolector de basura de Java borra el objeto, liberando así la memoria que ocupaba previniendo posibles fugas (ejemplo: un objeto creado y únicamente usado dentro de un método sólo tiene entidad dentro de éste; al salir del método el objeto es eliminado). Aún así, es posible que se produzcan fugas de memoria si el código almacena referencias a objetos que ya no son necesarios—es decir, pueden aún ocurrir, pero en un nivel conceptual superior. En definitiva, el recolector de basura de Java permite una fácil creación y eliminación de objetos, mayor seguridad y puede que más rápida que en C++.

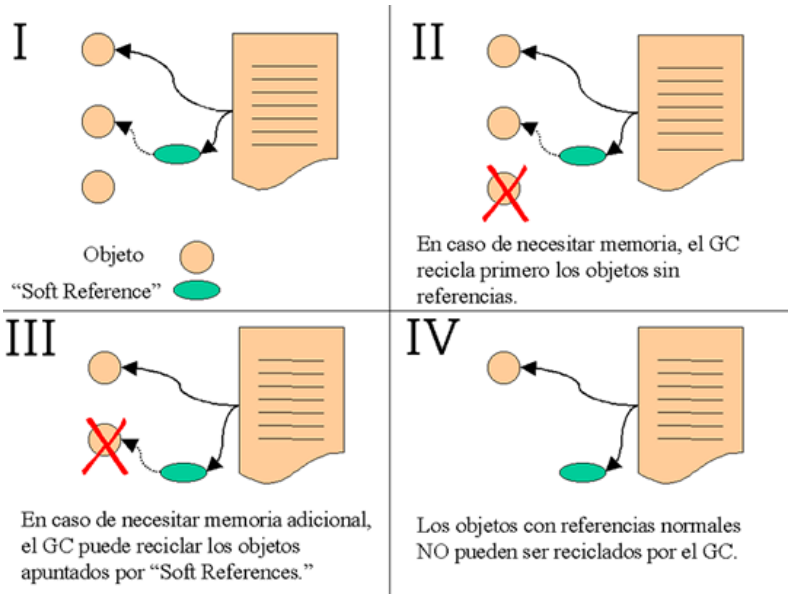


Figura 48.Funcionamiento del recolector de basura.

La recolección de basura de Java es un proceso prácticamente invisible al desarrollador. Es decir, el programador no tiene conciencia de cuándo la recolección de basura tendrá lugar, ya que ésta no tiene necesariamente que guardar relación con las acciones que realiza el código fuente.

Debe tenerse en cuenta que la memoria es sólo uno de los muchos recursos que deben ser gestionados.

3.1.6 Recursos de Java

JRE

El **JRE** (Java Runtime Environment, o Entorno en Tiempo de Ejecución de Java) es el software necesario para ejecutar cualquier aplicación desarrollada para la plataforma Java. El usuario final usa el JRE como parte de paquetes software o plugins (o conectores) en un navegador Web. Sun ofrece también el SDK de Java 2, o JDK (Java Development Kit) en cuyo seno reside el JRE, e incluye herramientas como el compilador de Java, Javadoc para generar documentación o el depurador. Puede también obtenerse como un paquete independiente, y puede considerarse como el entorno necesario para ejecutar una aplicación Java, mientras que un desarrollador debe además contar con otras facilidades que ofrece el JDK.

Componentes

- Bibliotecas de Java, que son el resultado de compilar el código fuente desarrollado por quien implementa la JRE, y que ofrecen apoyo para el desarrollo en Java. Algunos ejemplos de estas librerías son:
    - Las bibliotecas centrales, que incluyen:
        - Una colección de bibliotecas para implementar estructuras de datos como listas, arrays, árboles y conjuntos.
        - Bibliotecas para análisis de XML.
        - Seguridad.
        - Bibliotecas de internacionalización y localización.
    - Bibliotecas de integración, que permiten la comunicación con sistemas externos. Estas librerías incluyen:
        - La API para acceso a bases de datos JDBC (Java DataBase Conectivity).
        - La interfaz JNDI (Java Naming and Directory Interface) para servicios de directorio.
        - RMI (Remote Method Invocation) y CORBA para el desarrollo de aplicaciones distribuidas.

- Bibliotecas para la interfaz de usuario, que incluyen:
  - El conjunto de herramientas nativas AWT (Abstract Windowing Toolkit), que ofrece componentes GUI (Graphical User Interface), mecanismos para usarlos y manejar sus eventos asociados.
  - Las Bibliotecas de Swing, construidas sobre AWT pero ofrecen implementaciones no nativas de los componentes de AWT.
  - APIs para la captura, procesamiento y reproducción de audio.

- Una implementación dependiente de la plataforma en que se ejecuta de la máquina virtual de Java (JVM), que es la encargada de la ejecución del código de las librerías y las aplicaciones externas.
- Plugins o conectores que permiten ejecutar applets en los navegadores Web.
- Java Web Start, para la distribución de aplicaciones Java a través de Internet.
- Documentación y licencia.

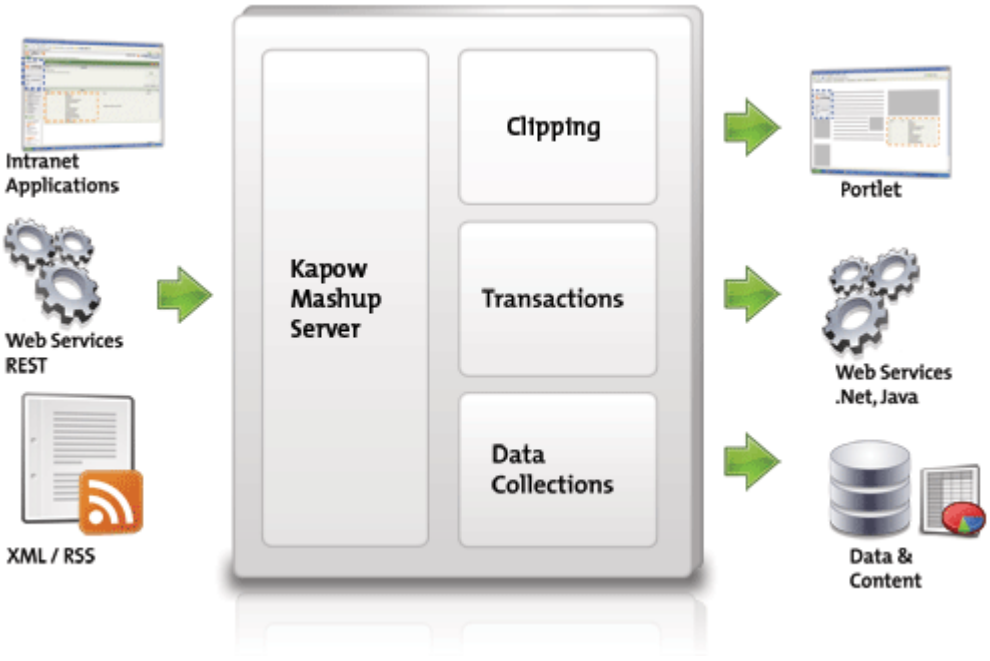


Figura 49. Esquema de los componentes de Java

APIs

Sun define tres plataformas en un intento por cubrir distintos entornos de aplicación. Así, ha distribuido muchas de sus APIs (Application Program Interface) de forma que pertenezcan a cada una de las plataformas:

- Java ME (Java Platform, Micro Edition) o J2ME — orientada a entornos de limitados recursos, como teléfonos móviles, PDAs (Personal Digital Assistant), etc.
- Java SE (Java Platform, Standard Edition) o J2SE — para entornos de gama media y estaciones de trabajo. Aquí se sitúa al usuario medio en un PC de escritorio.
- Java EE (Java Platform, Enterprise Edition) o J2EE — orientada a entornos distribuidos empresariales o de Internet.

Las clases en las APIs de Java se organizan en grupos disjuntos llamados paquetes. Cada paquete contiene un conjunto de interfaces, clases y excepciones relacionadas. La información sobre los paquetes que ofrece cada plataforma puede encontrarse en la documentación de ésta.

El conjunto de las APIs es controlado por Sun Microsystems junto con otras entidades o personas a través del programa JCP (Java Community Process). Las compañías o individuos participantes del JCP pueden influir de forma activa en el diseño y desarrollo de las APIs, algo que ha sido motivo de controversia.

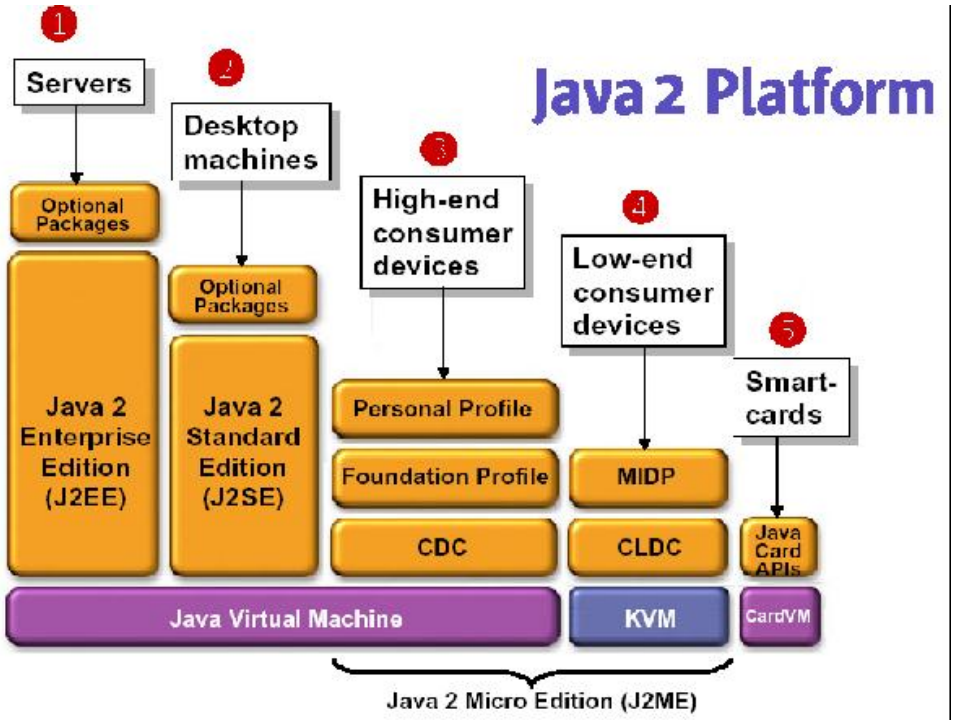


Figura 50. API de Java

3.2 J2ME (Java 2 Micro Edition)

J2ME (Java 2 Micro Edition) es la plataforma basada en el lenguaje Java que Sun Microsystems ha creado para la programación de dispositivos inalámbricos pequeños como teléfonos celulares, paginadores y PDA. La siguiente figura muestra como está compuesta la plataforma J2ME.

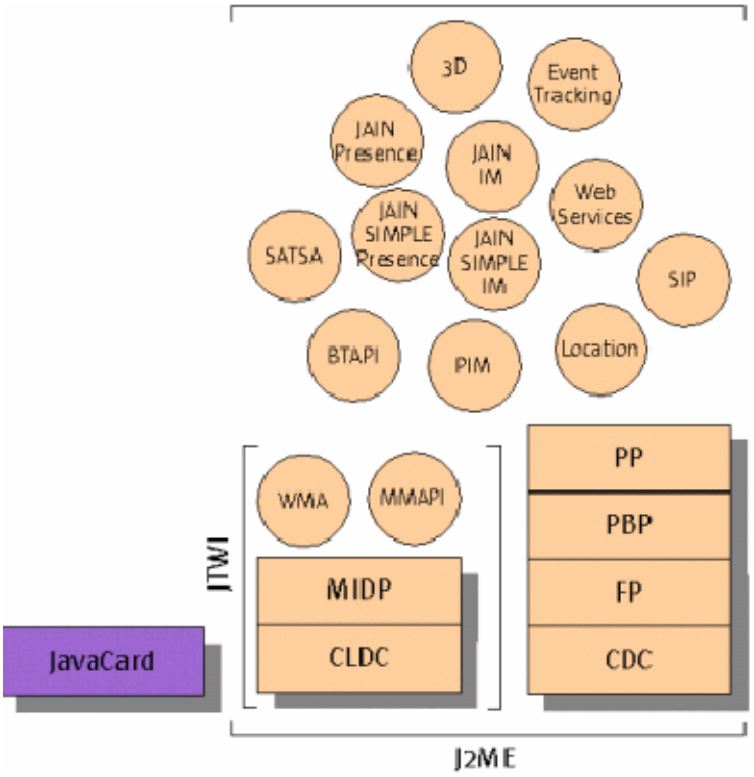


Figura 51. Esquema de la arquitectura J2ME

J2ME contiene una mínima parte de las APIs de Java. Esto es debido a que la edición estándar de APIs de Java ocupa 20 MB, y los dispositivos pequeños disponen de una cantidad de memoria mucho mas reducida. En concreto, J2ME usa 37 clases de la plataforma J2SE provenientes de los paquetes java.lang, java.io, java.util. Esta parte de la API que se mantiene fija forma parte de lo que se denomina configuración, ya se hablara de ella más ampliamente en la siguiente subsección. Otras diferencias con J2SE vienen dadas por el uso de una maquina virtual distinta de la clásica JVM denominada KVM. Esta KVM tiene unas restricciones que hacen que no posea todas las capacidades incluidas en la JVM. Estas diferencias se verán más detenidamente cuando analicemos la KVM.

Como se puede ver, J2ME representa una versión simplificada de J2SE. Sun separó estas dos versiones ya que J2ME estaba pensada para dispositivos con limitaciones de proceso y capacidad gráfica. Se puede afirmar que J2ME es un subconjunto de J2SE exceptuando el paquete javax.microedition perteneciente a J2ME y no existente en J2SE.

Solo de una manera muy simplista se puede considerar a J2ME y J2EE como versiones reducidas y ampliadas de J2SE respectivamente, ya que en realidad cada una esta enfocada a ámbitos totalmente diferentes y por ello difieren sus capacidades. Las necesidades computacionales y APIs de programación requeridas para un juego ejecutándose sobre un dispositivo móvil son totalmente distintas de las que requiere un servidor distribuido de aplicaciones basado en EJB.

3.2.1 Perfiles, configuraciones y máquinas virtuales

La edición micro de Java se compone, además del lenguaje, de una máquina virtual, configuraciones, perfiles y paquetes adicionales.

La máquina virtual es la base de la plataforma, es el intérprete del lenguaje y sobre la cual se han de ejecutar las aplicaciones, también sobre esta máquina virtual corren las configuraciones (CDC y CLDC), las cuales incorporan Apis básicas para la creación de aplicaciones y sirven de soporte a los perfiles. Los perfiles incluyen la mayor parte de las clases y Apis que se van a utilizar en la programación, como pueden ser instrucciones de entrada y salida o de inicio y terminación de la aplicación.

Los paquetes adicionales son aquellos que la especificación de la tecnología inalámbrica Java (JSR185) no establece como obligatorios para incorporar en los dispositivos. Ejemplos de esto pueden ser las Apis de mensajes inalámbricos (WMAPI) y de multimedia (WMAPI).

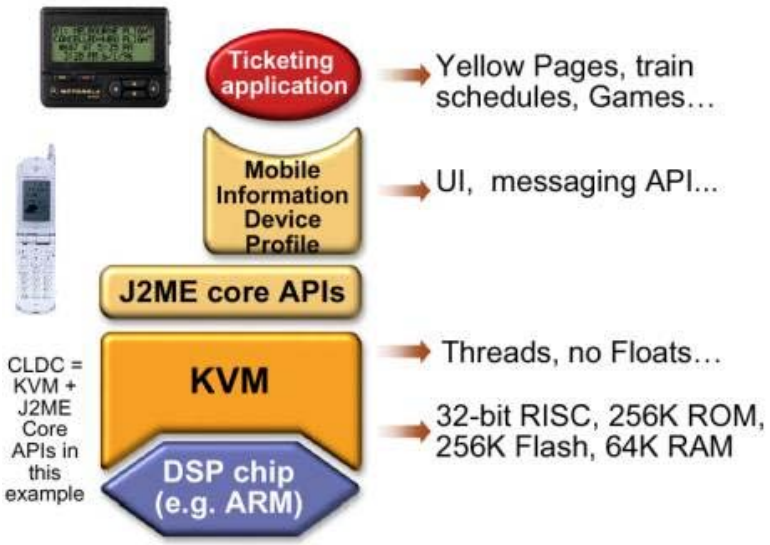


Figura 52. Perfiles, configuraciones y máquinas virtuales Java.

3.2.2 Maquinas virtuales de J2ME

KVM (Kilobyte Virtual Machine)

Como ya se explicó anteriormente la máquina virtual es la base de la plataforma Java.

En el caso de la plataforma J2ME, debido a las capacidades limitadas de almacenamiento, memoria, procesamiento y pantalla de los dispositivos; no se puede integrar una máquina virtual Java (JVM) del las dimensiones de J2SE o J2EE. Por esto se ha creado una nueva máquina virtual: KVM (Kilobyte Virtual Machine)

KVM es una máquina virtual Java compacta y portable específicamente diseñada para ser la base de desarrollo en dispositivos pequeños y de recursos limitados. Actualmente CLDC trabaja sobre KVM. Además KVM está diseñada para mantener los aspectos centrales del lenguaje Java ejecutándose en unos cuantos kilobytes de memoria (de ahí su nombre).

Aunque KVM deriva de la máquina virtual J2SE (JVM), algunas características de esta última han sido eliminadas para soportar CLDC, debido a que resultan demasiado costosas de implementar o su presencia supone problemas de seguridad, resultando una KVM con las siguientes limitantes:

- Soporte de punto flotante. KVM no soporta números de punto flotante, esto debido a que la mayoría de los dispositivos en los que se implementa no lo soportan tampoco.

- Finalización. Las APIs CLDC no incluyen el método *object.finalize*, así que no se pueden hacer operaciones de limpieza final antes de que el recolector de basura tome los objetos.
- Manejo de errores. CLDC sólo define tres clases error: *java.lang.Error*, *java.lang.OutOfMemoryError* y *java.lang.VirtualMachineError*. Todo tipo de errores que no sean en tiempo de ejecución se manejan de modo dependiente del dispositivo, esto incluye la finalización de una aplicación o reinicio del dispositivo.

- Interfaz Nativa Java (JNI). No se implementa JNI (posibilidad de incluir código en C dentro de clases Java), primeramente por motivos de seguridad, aunque también es considerado excesivo dadas las limitantes de memoria del dispositivo al que se dirige.

- Cargadores de clases definidas por el usuario. KVM debe tener un cargador de clases que no pueda ser manipulado o remplazado por el usuario, principalmente por razones de seguridad.

- No hay soporte para la API *reflection*. La API *reflection* es más bien usada para aplicaciones de bajo nivel (depuradores, constructores GUI, etc). La falta de esta API impide también la serialización de objetos.

- Grupos de hilos o hilos "demonio". Aunque CLDC soporta programación multihilo, no soporta grupos de hilos o hilos "demonio". Si requiere usar operaciones para grupos de hilos use objetos de colección para almacenar los objetos hilo a nivel de aplicación.

- Referencias débiles. Ninguna aplicación construida con CLDC puede requerir referencias débiles.

El verificador de clases estándar de Java es demasiado grande para la KVM. De hecho es mas grande que la propia KVM y el consumo de memora es excesivo, mas de 100 Kb para las aplicaciones típicas. Este verificador de clases es el encargado de rechazar las clases no válidas en tiempo de ejecución. Este mecanismo verifica los bytecodes de las clases Java realizando las siguientes comprobaciones:

- Comprueba que el código no sobrepase los límites de la pila de la VM.
- Comprueba que no se utilicen variables locales antes de ser inicializadas.
- Comprueba que se respeten los campos, métodos y los modificadores de control de acceso a clases.

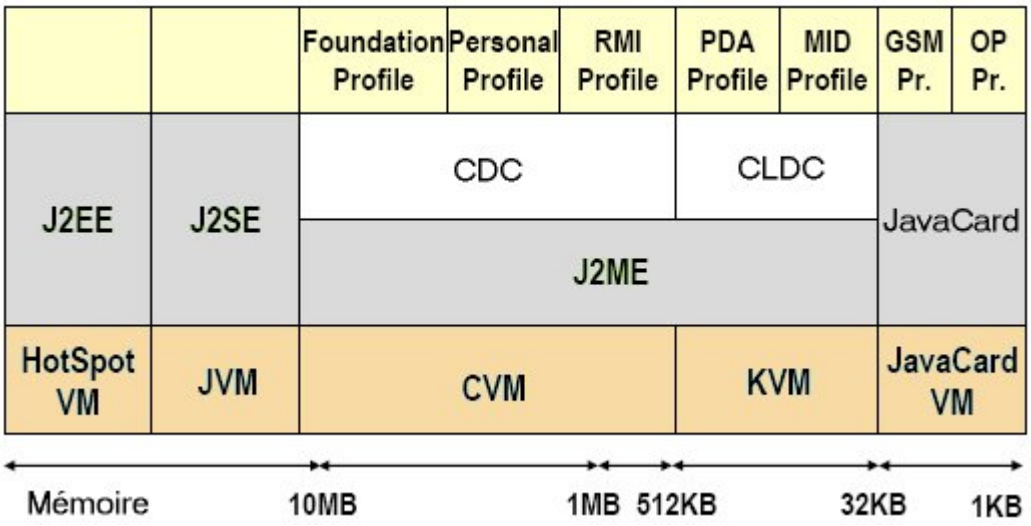


Figura 53. Arquitectura Java

Los diseñadores de KVM decidieron mover la mayor parte del trabajo de verificación de clases fuera del dispositivo, es decir, hacia la computadora de escritorio donde las clases son compiladas o el servidor de donde se descargan. A este proceso de verificación se le llama preverificación. Los dispositivos son únicamente responsables de

ejecutar las pruebas anteriormente nombradas en la clase preverificada para asegurarse de que aún es válida.

Por este motivo los dispositivos que usen la configuración CLDC y KVM introducen un algoritmo de verificación de clases de dos pasos.

La KVM puede ser compilada y probada en 3 plataformas diferentes:

1. Solaris Operating System
2. Windows
3. PalmOs

CVM (Compact Virtual Machine)

La CVM ha sido tomada como Máquina Virtual Java de referencia para la configuración CDC y soporta las mismas características que la maquina virtual de J2SE. Esta orientada a dispositivos electrónicos con procesadores de 32 bits de gama alta y en torno a 2 Mb o mas de memoria RAM. Las características que presenta esta Máquina Virtual son:

- Sistema de memoria avanzado.
- Tiempo de espera bajo para el recolector de basura.
- Portabilidad.
- Baja ocupación en memoria de las clases.
- Separación completa de la VM del sistema de memoria.
- Recolector de basura modularizado.
- Ejecución de las clases Java fuera de la memoria de solo lectura (ROM).
- Conversión de hilos Java a hilos nativos.
- Soporte nativo de hilos.
- Rápida sincronización.
- Proporciona soporte e interfaces para servicios en Sistemas Operativos de Tiempo Real.
- Soporte para todas las características de Java2 v1.3 y librerías de seguridad, referencias débiles, Interfaz Nativa de Java (JNI), invocación remota de métodos (RMI), Interfaz de depuración de la Maquina Virtual (JVMDI).

3.2.3 Configuraciones

Una configuración es el conjunto mínimo de APIs Java que permiten desarrollar aplicaciones para un grupo de dispositivos. En J2ME existen dos configuraciones, CLDC, orientada a dispositivos con limitaciones computacionales y de memoria, CDC, orientada a dispositivos con menos limitaciones. Ahora se verán con más profundidad cada una de estas configuraciones:

CDC, configuración de dispositivos con conexión (Conected Limited Configuration)

Esta orientada a dispositivos con cierta capacidad computacional y de memoria. Por ejemplo, electrodomésticos, sistemas de navegación de automóviles y decodificadores de televisión. CDC como ya se mencionó antes utiliza una Máquina Virtual de Java similar a

la de J2SE, pero con algunas limitaciones en el apartado gráfico y de memoria del dispositivo. Esta maquina virtual es la que se ha visto como CVM. La CDC esta enfocada a dispositivos con las siguientes capacidades:

- Procesador de 32 bits.
- Disponer de 2 MB o más de memoria total, incluyendo memoria RAM y ROM.
- Poseer la funcionalidad completa de la Máquina Virtual Java2.
- Conectividad a algún tipo de red.

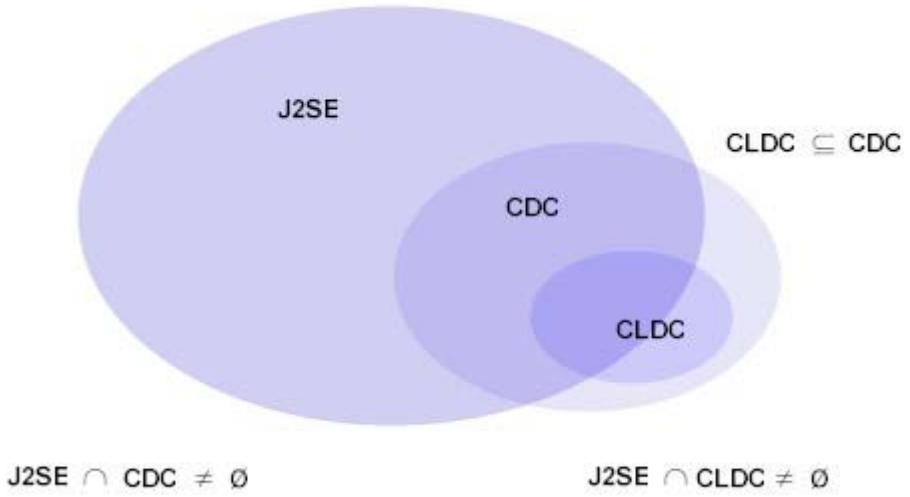


Figura 54. Diagrama de la relación entre J2SE y las configuraciones CDC y CLDC

La CDC esta basada en J2SE v1.3 e incluye varios paquetes Java de la edición estándar. La peculiaridad de CDC esta básicamente en el paquete javax.microedition.io, que incluye soporte para comunicaciones http y basadas en datagramas. Las librerías incluidas en la CDC son las siguientes:

- java.io
- java.lang
- java.lang.ref
- java.lang.reflect
- java.math
- java.net
- java.security
- java.security.cert
- java.text
- java.util
- javax.microedition.io

CLDC, configuración de dispositivos limitados con conexión (Connected Limited Device Configuration)

La CLDC esta orientada a dispositivos dotados de conexión pero con limitaciones en cuanto a capacidad gráfica, computacional y memoria. Unos ejemplos de estos tipos de dispositivos son los teléfonos móviles, las PDAs y los organizadores personales. CLDC es la base para que los perfiles (como MIDP o PDAP) funcionen, proveyendo las Apis básicas y la máquina virtual (KVM). CLDC está diseñada para equipos microprocesadores RISC o CISC de 16 a 32 bits y con una memoria mínima de 160 KB para la pila de la tecnología Java.

La JSR185 pide como requisitos mínimos para todo equipo que implemente CLDC 1.0 o 1.1:

- Soporte mínimo para diez hilos relacionados con aplicaciones (MIDlets).
- Usar zonas de tiempo personalizables, con referencia en el formato de zonas de tiempo GMT (GMT \7:00, por ejemplo).
- Soporte para propiedades de carácter y conversiones mayúsculas-minúsculas en los bloques supleméntales Unicode para Basic Latin y Latin-1 (nuestros caracteres).

La CLDC aporta las siguientes funcionalidades a los dispositivos:

- Un subconjunto del lenguaje Java y todas las restricciones de su Máquina Virtual (KVM).
- Un subconjunto de las bibliotecas Java del núcleo.
- Soporte para E/S básica.
- Soporte para acceso a redes.
- Seguridad.

Las librerías incluidas en CLDC son las siguientes:

- java.lang
- java.io
- java.util
- javax.microedition.io

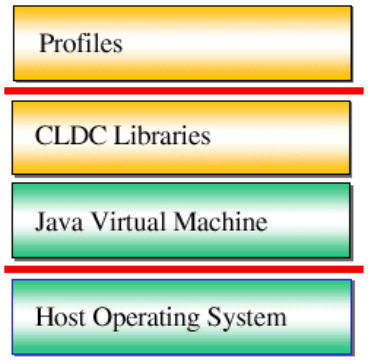


Figura 55. CLDC

Las APIs para CLDC tienen el objetivo de proveer un conjunto de bibliotecas mínimo y útil para el desarrollo de aplicaciones y definición de perfiles para una gran variedad de aparatos.

Las APIs CLDC se pueden dividir en dos categorías:

- Clases derivadas de APIs J2SE. Estas se localizan en los paquetes *java.lang*, *java.io* y *java.util* y se derivan de APIs del Java Development Kit (JDK) 1.3.
- Clases específicas para CLDC. Se localizan en el paquete *javax.microedition* y sus subpaquetes.

Clases heredadas (derivadas): CLDC hereda algunas clases de sistema, entrada y salida (E/S) y utilidades de la plataforma J2SE.

Clases de NO excepción heredadas de J2SE:

Paquete	*Clases*
java.lang	Boolean, Byte, Character, Class, Integer, Long, Math, Object, Runnable, Runtime, Short, String, StringBuffer, System, Thread, Throwable
java.io	ByteArrayInputStream, ByteArrayOutputStream, DataInput, DataOutput, DataInputStream, DataOutputStream, InputStream, OutputStream, InputStreamReader, OutputStreamWriter, PrintStream, Reader, Writer
java.util	Calendar, Date, Enumeration, Hashtable, Random, Stack, TimeZone, Vector

Clases de excepción heredadas de J2SE:

Paquete	*Clases*
java.lang	ArithmeticException, ArrayIndexOutOfBoundException, ArrayStoreException, ClassCastException, ClassNotFoundException, Error, Exception, IllegalAccessException, IllegalArgumentException, IllegalMonitorStateException, IllegalThreadStateException, IndexOutOfBoundException, InstantiationException, InterruptedException, OutOfMemoryError, NegativeArraySizeException, NumberFormatException, NullPointerException, RuntimeException, SecurityException, StringIndexOutOfBoundException, VirtualMachineError
java.io	EOFException, IOException, InterruptedException, UnsupportedEncodingException, UTFDataFormatException
java.util	EmptyStackException, NoSuchElementException

En CLDC no hay implementación para la clase *java.util.Properties*, sin embargo las propiedades mostradas en la siguiente tabla están disponibles y se pueden obtener llamando al método *System.getProperty(clave),* donde clave puede ser cualquiera de las siguientes:

Clave	*Explicación*	*Valor predeterminado*
microedition.platform	La plataforma o dispositivo huésped.	null
microedition.encoding	Codificación predeterminada de caracteres.	ISO8859_1
microedition.configurations	Configuración y versión J2ME actual.	CLDC-1.0
microedition.profiles	Nombre de los perfiles soportados.	null

Clases específicas de CLDC: Las siguientes clases son específicas de CLDC y están contenidas en el paquete *javax.microedition.io*:

Paquete	*Clases*
javax.microedition.io	Connection, ConnectionNotFoundException, Connector, ContentConnector, Datagram, DatagramConnection, InputConnection, OutputConnection, StreamConnection, StreamConnectionNotifier

Un aspecto muy a tener en cuenta es la seguridad en CLDC. Esta configuración posee un modelo de seguridad sandbox (cajón de arena) al igual que ocurre con los applets que proporcionaba un entorno muy restringido en el que ejecutar código no fiable obtenido de la red.

En este modelo se trabaja con dos niveles de acceso a los recursos: total, para programas locales, o muy restringido, para programas remotos. La seguridad se consigue gracias al empleo del cargador de clases, el verificador de clases y el gestor de seguridad.

La pega fundamental de este modelo es que es demasiado restrictivo, ya que no permite que los programas remotos hagan nada útil, por estar restringidos al modelo del cajón de arena.

Hay una versión mejorada de este sistema que consiste en un sistema de control de permisos de grano fino, el cual permite dar permisos específicos a trozos de código específicos para acceder a recursos específicos en el cliente, dependiendo de la firma del código y/o el URL del que este se obtuvo.

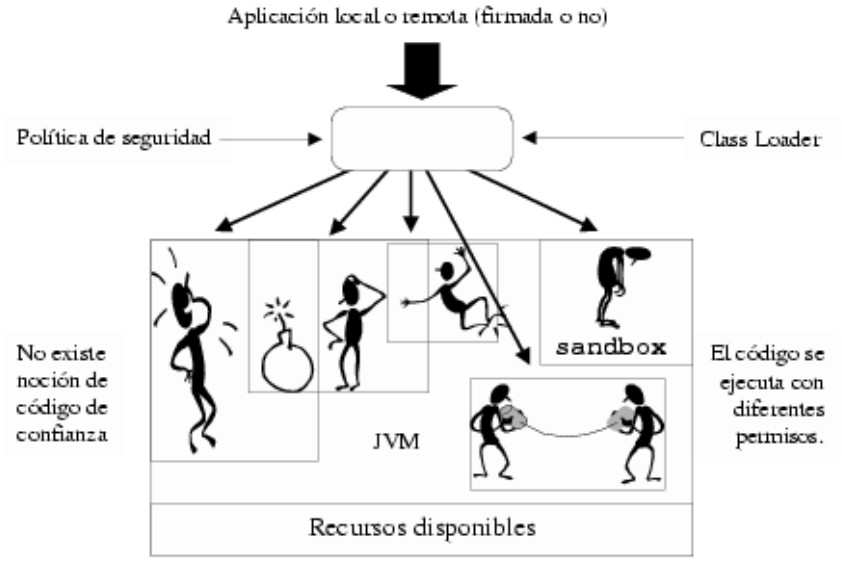


Figura 56. Ilustración de los recursos de seguridad en J2ME

3.2.4 Perfiles en J2ME

El perfil es el que define las APIs que controlan el ciclo de vida de la aplicación, interfaz de usuario, etc. Más concretamente, un perfil es un conjunto de APIs orientado a un amito de aplicación determinado. Los perfiles identifican un grupo de dispositivos por la funcionalidad que proporcionan y el tipo de aplicaciones que se ejecutarán en ellos. Las librerías de la interfaz gráfica son un componente muy importante en la definición de un perfil.

El perfil establece unas APIs que definen características de un dispositivo, mientras que la configuración hace lo propio con una familia de ellos. Esto hace que a la hora de construir una aplicación se cuente tanto con las APIs del perfil como de la configuración. Hay que tener en cuenta que un perfil siempre se construye sobre una configuración determinada. De este modo, podemos pensar en un perfil como un conjunto de APIs que dotan a una configuración de funcionalidad especifica.

Anteriormente se vio que para una configuración determinada se usaba una Maquina Virtual Java específica. Con los perfiles ocurre algo similar, existen unos perfiles que se usarán sobre la configuración CDC y otros sobre la CLDC. Para la configuración CDC existen los siguientes perfiles:

- Foundation Profile.
- Personal Profile.
- RMI Profile.

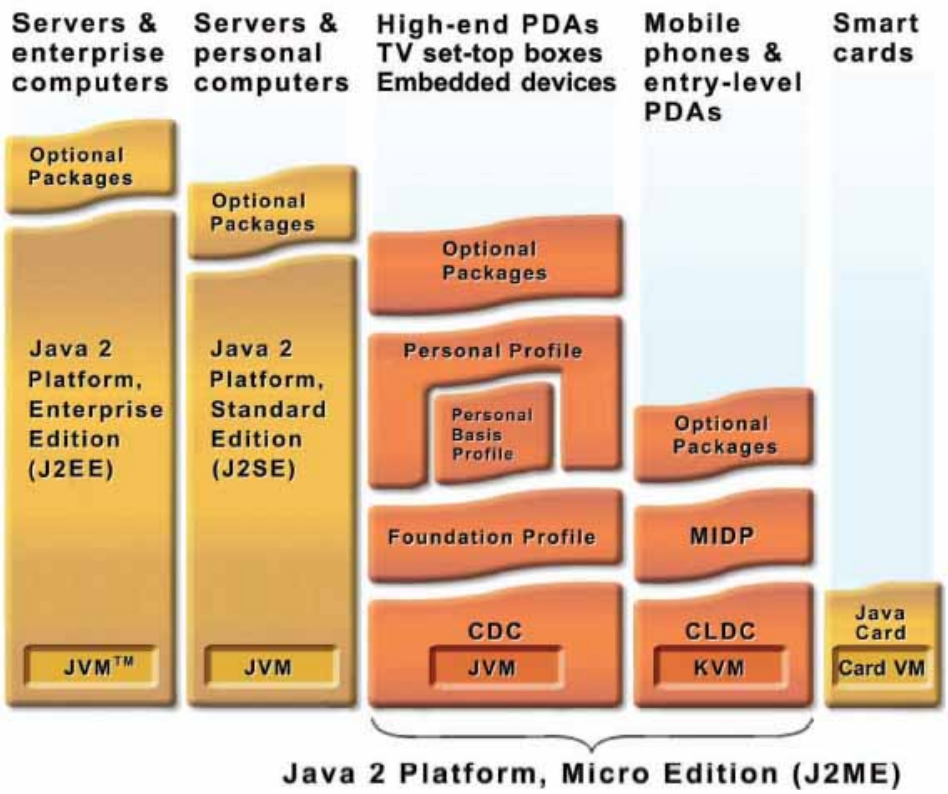


Figura 57. Esquema de los perfiles en J2ME

Y para la configuración CLDC:

- PDA Profile.
- Mobile Information Device Profile (MIDP).

Un perfil puede ser construido sobre cualquier otro. Sin embargo, una plataforma J2ME sólo puede contener una configuración.

A continuación se explicarán los diferentes perfiles más en profundidad.

- *Foundation Profile:* Este perfil define una serie de APIs sobre la CDC orientadas a dispositivos que carecen de interfaz gráfica, como por ejemplo los decodificadores de televisión digital. Este perfil incluye gran parte de los paquetes de la J2SE, pero excluye totalmente los paquetes "java.awt" (Abstract Windows Toolkit, AWT) y "java.swing" que conforman la interfaz gráfica de usuario (GUI) de J2SE. Si una aplicación requiriese una GUI, entonces seria necesario un perfil adicional. Los paquetes que forman parte del Foundation Profile son los siguientes:

    1. *Java.lang*
    2. *Java.util*
    3. *Java.net*
    4. *Java.io*
    5. *Java.text*
    6. *Java.security*

- *Personal Profile:* El Personal Profile es un subconjunto de la plataforma J2SE v1.3, y proporciona un entorno con un completo soporte gráfico AWT. El objetivo es el de dotar a la configuración CDC de una interfaz gráfica completa, con capacidades web y soporte de applets Java. Este perfil requiere la implementación del Foundation Profile. Los paquetes que conforman el Personal Profile v1.0 son:

    1. *Java.applet*
    2. *Java.awt*
    3. *Java.awt.datatransfer*
    4. *Java.awt.event*
    5. *Java.awt.font*
    6. *Java.awt.im*
    7. *Java.awt.im.spi*
    8. *Java.awt.image*
    9. *Java.beans*
    10. *Javax.microedition.xlet*

- *RMI Profile:* Este perfil requiere la implementación del Foundation Profile, se construye encima de el. El perfil RMI soporta un subconjunto de las APIs v1.3 RMI. Algunas características de estas APIs se han eliminado del perfil RMI debido

a las limitaciones de cómputo y memoria de los dispositivos. Las siguiente propiedades se han eliminado del J2SE RMI v1.3:

1. *Java.rmi.server.disableHTTP.*
2. *Java.rmi.activation.port.*
3. *Java.rmi.loader.packagePrefix.*
4. *Java.rmi.registry.packagePrefix.*
5. *Java.rmi.server.packagePrefix.*

- *PDA Profile:* El PDA Profile esta construido sobre CLDC. Pretende abarcar PDAs de gama baja, tipo Palm, con una pantalla y algún tipo de puntero (ratón o lápiz) y una resolución de al menos 20000 pixels (al menos 200x100 pixels) con un factor 2:1.

- *Mobile Information Device Profile (MIDP):* Construido sobre la configuración CLDC. MIDP fue el primer perfil definido para esta plataforma. Hasta este momento es el único perfil aplicado a los dispositivos en el mercado, aunque se están investigando algunos otros, como el especializado en PDA. MIDP 2.0 incorpora APIs de interfaz de usuario, de ciclo de vida del programa, almacenamiento persistente, juegos, trabajo en red y multimedia. Según la especificación de la tecnología inalámbrica de Java todo dispositivo que soporte MIDP 2.0 debe incluir mínimamente las siguientes características:

  1. Debe permitir archivos Java (JAR) de más de 64 KB. y archivos descriptores de aplicaciones (JAD) mayores a 5 KB.
  2. Se debe permitir a cada MIDlet la utilización de 30 KB de almacenamiento persistente y se recomienda que las MIDlets incluyan información acerca del almacenamiento mínimo con el que trabajan correctamente.
  3. El espacio de memoria libre para una aplicación ejecutándose (Heap o del montón) debe ser por lo menos de 256 KB.
  4. Soporte para pantallas de 125 x 125 pixeles, con una profundidad de color de 12 bits.
  5. Se deben incluir la capacidad de que el dispositivo reaccione a eventos de tiempo (una alarma a determinada hora, los llamados ticklers o despertadores).
  6. Mecanismos para tomar un número telefónico del directorio del equipo.
  7. Soporte para imágenes en formato JPEG y PNG.
  8. Acceso a contenidos multimedia por el protocolo HTTP 1.1.

Los paquetes que están incluidos en MIDP son los siguientes:

- Javax.microedition.lcdui
- Javax.microedition.rms
- Javax.microedition.midlet
- Javax.microedition.io
- Java.io

- Java.lang
- Java.util

Además de las clases específicas de MIDP contenidas en *javax.microedition.rms*, *javax.microedition.midlet* y *javax.microedition.lcdui*; están disponibles las siguientes clases, interfaces y clases de excepción:

- *IllegalStateException.* Clase en el paquete *java.lang*.
- *Timer* y *TimerTask.* Clases en el paquete *java.util*.
- *HttpConnection.* Interfaz para acceso a una red por el protocolo HTTP contenida en el paquete *javax.microedition.io*

Las aplicaciones creadas utilizando MIDP reciben el nombre de MIDlets. Se puede decir que un MIDlet es una aplicación Java realizada con el perfil MIDP sobre la configuración CLDC.

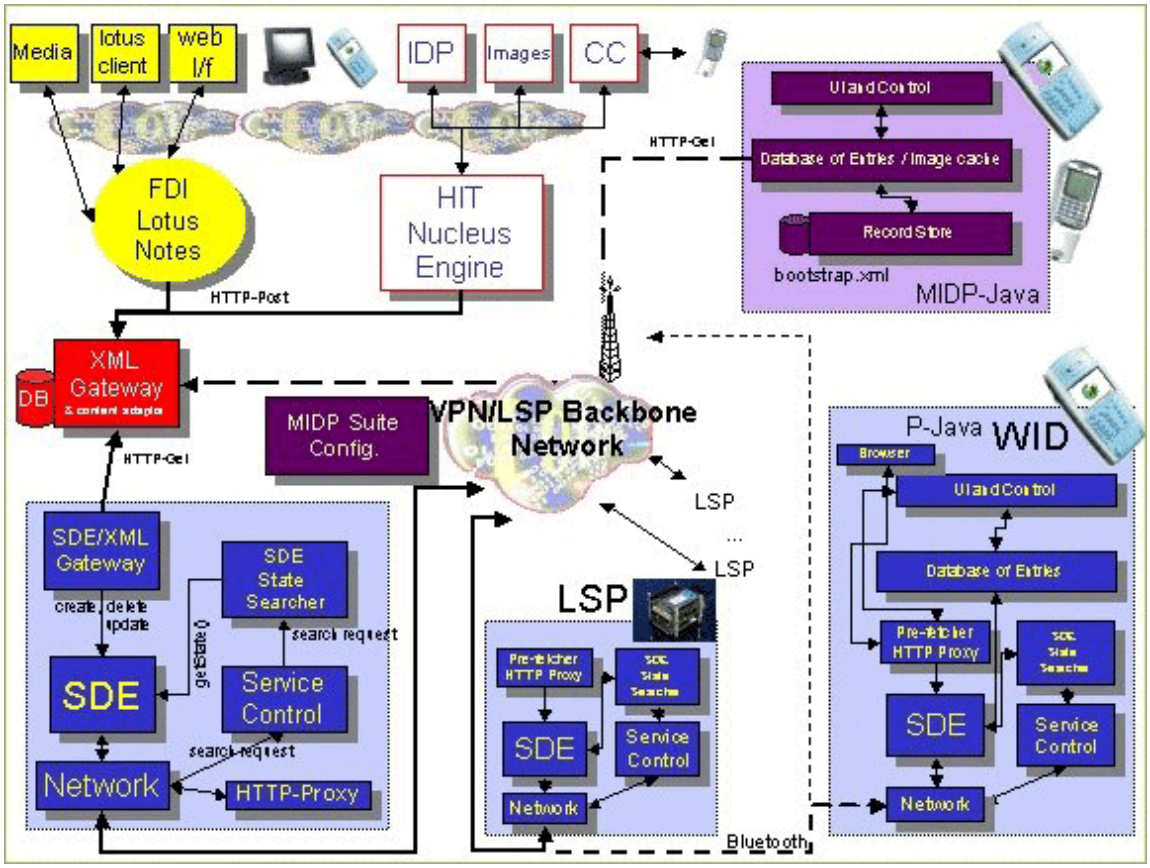


Figura 58. Esquema de funcionalidades del P800 en J2ME

3.3 Desarrollo de aplicaciones y uso de herramientas

En esta parte del proyecto se van a desarrollar un conjunto de aplicaciones en Java, se intentará mostrar el uso eficiente de todas las herramientas proporcionadas por Symbian para este lenguaje. Se estructurará en 5 capítulos pertenecientes cada uno a un ámbito determinado del sistema. Todas ellas se realizarán sobre una configuración CLDC (adecuada para dispositivos móviles) y un perfil MIDP.

3.3.1 Introducción a Herramientas y entorno Symbian

En este primer capitulo se hará una introducción a la instalación y manejo de todas las herramientas necesarias para crear una aplicación en Symbian OS. El proceso de creación de una aplicación puede ser básicamente de dos maneras:

- A través de la línea de comandos: para este caso no se utilizará ninguna herramienta específica Symbian, simplemente se crearan los códigos necesarios y se compilaran con el compilador adecuado.
- A través de un IDE: un IDE es un entorno gráfico diseñado para facilitar las funciones de un programador de aplicaciones Symbian. Estos entornos suelen integrar todas las herramientas necesarias, entre ellas los compiladores, debugadores y emuladores que disminuirán la dificultad para crear aplicaciones.

Antes de empezar a explicar los pasos a seguir para instalar las aplicaciones necesarias, es importante repasar las etapas que hay que cumplimentar en la creación de cualquier tipo de programa:

- Escritura: En esta fase se escribirá el código que conforma nuestra aplicación.
- Compilación: Una vez escrito el código, este se compilará con un compilador J2ME.
- Preverificación: Antes de empaquetar el código realizaremos un proceso de preverificación sobre el mismo. En esta fase se realiza un examen del código para confirmar que no viola ninguna restricción de seguridad de J2ME.
- Empaquetamiento: En esta fase se creará un archivo JAR para poder probar la aplicación en un emulador. También se creará un archivo JAD para instalar la aplicación en el dispositivo.
- Ejecución: Haciendo uso de los emuladores se comprobará el correcto funcionamiento de nuestra aplicación en el pc.
- Depuración: Esta es una fase típica de la realización de cualquier tipo de aplicación en cualquier tipo de lenguaje. Se depurarán los fallos detectados en la etapa anterior, pudiendo hacer uso para esta de los debugadores.

Cualquier tipo de aplicación en Java sigue este proceso, exceptuando las etapas de empaquetamiento y preverificación que son especificas de J2ME.

3.3.1.1 Desarrollo en línea de comandos:

Para empezar se verá como desarrollar aplicaciones MIDP utilizando solo la línea de comandos. Para esta primera iniciación se utilizaran las siguientes herramientas:

- Editor de texto: se puede utilizar cualquier editor que haya en el ordenador. Desde el Bloc de notas de Windows al Emacs o Vi de Linux.
- Compilador: Un compilador estándar de Java. Haciendo uso del SDK de J2SE. Este se puede descargar desde la página oficial de Sun. http://developers.sun.com/downloads
- CLDC: Las APIs de la configuración CLDC y del perfil MIDP. Se pueden encontrar en el siguiente enlace. http://java.sun.com/products/sjwtoolkit/download.html

Una vez claras las herramientas necesarias y descargadas de los sitios web pertinentes, se pasará a la instalación del SDK de J2SE, para ello se hará doble click sobre el archivo descargado e instalarlo en una carpeta de nuestro disco duro (p.e. c:\jdk). Una vez instalado se deberá añadir el path de la carpeta anterior. Para ello se modificara la variable de entorno path a la que se añadirá la carpeta c:\jdk\bin y es necesario crear una variable de entorno llamada JAVA_HOME con el valor c:\jdk. Esto es necesario hacerlo con la carpeta en cuestión donde ha sido instalado el jdk.

Una vez instalado el jdk es el momento de instalar las APIs de CLDC y de MIDP. Hoy por hoy desde la dirección web apuntada anteriormente se descarga todo a la vez, así que solo es necesario hacer doble click sobre el archivo sun_java_wireless_toolkit-2_5_1-windows.exe (porque esta configuración es para Windows) y se instalará en la carpeta que sea elegida (p.e. c:\cldc). Al igual que con el jdk se deberá añadir la dirección c:\cldc\bin a la variable path y habrá que crear otra variable de entorno llamada MIDP_HOME con el valor c:\cldc.

Para comprobar que se ha realizado correctamente la instalación se deberá abrir una línea de comandos y escribir lo siguiente:

java –versión

Por la pantalla deberá aparecer algo como lo siguiente:

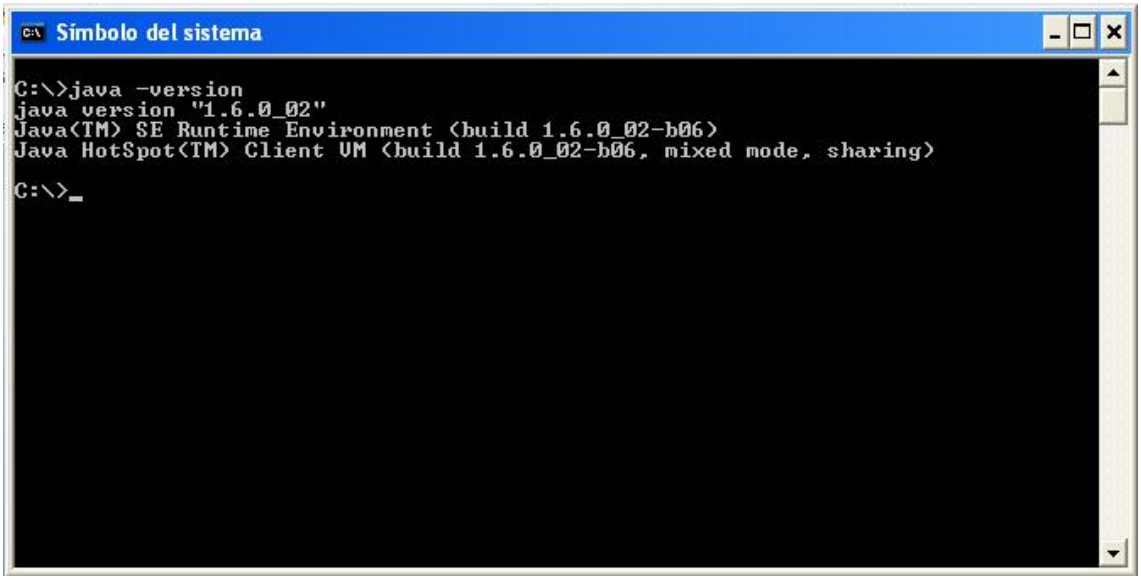


Figura 59. Captura de pantalla del símbolo del sistema durante la comprobación de la instalación del JSDK.

De esta manera se comprueba la correcta instalación del JSDK.

Para la comprobación de la correcta instalación del perfil y la configuración, es necesario escribir:

midp –version

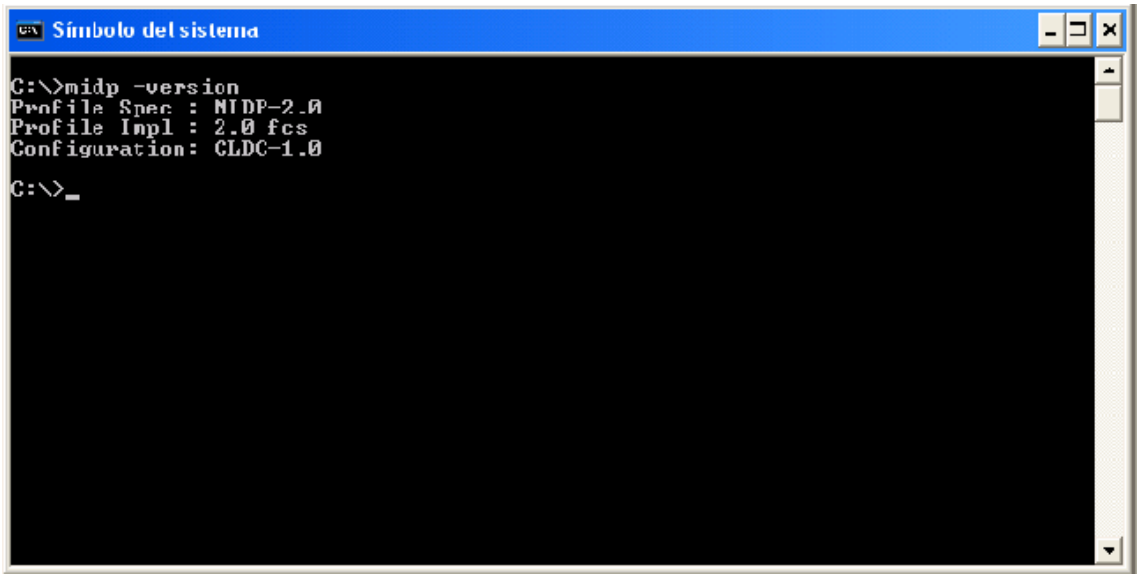


Figura 60. Captura de pantalla para la comprobación de la instalación del perfil MIDP

Una vez que esta instalado el compilador y el perfil se puede pasar al desarrollo y preparación de una aplicación. Esta se divide principalmente en cuatro fases:

1. *Desarrollo de código:* Haciendo uso de un editor de texto se escribirá todo el código de la aplicación. Es necesario guardar el fichero con el mismo nombre que la aplicación y con extensión .java (esto es básico en java, la clase ha de llamarse como el archivo porque si no, no compilará).
2. *Compilación:* En esta etapa se creará el archivo .class asociado a la clase .java. Para realizar la compilación habrá que abrir una línea de comandos y desplazarse hasta el directorio donde se encuentre el fichero <fuente>.java. En este punto se deberá escribir lo siguiente en la línea de comandos:

   javac –boothclasspath c:\cldc\classes <fuente>.java

3. *Preverificación:* En esta fase se realizara la preverificación de clases. Para ello habrá que moverse al directorio donde se encuentre la clase ya compilada y escribir:

   preverify –classpath c:\cldc\classes <fuente>.java

Por defecto la preverificación generara un fichero .class en el directorio ./output/.

4. *Empaquetamiento:* Ahora se empaquetará la aplicación para dejarla totalmente preparada con el fin de poder descargarla sobre el dispositivo MID (Mobile Information Device). Para ello se deberán construir dos archivos:

    a. Un archivo JAR con los ficheros que forman la aplicación (también se le puede llamar MIDlet).
    b. Un archivo descriptor de la aplicación que es opcional.

    Normalmente cuando se realiza un empaquetamiento están involucrados varios MIDlets que conforman lo que se denomina una suite de MIDlets. Para simplificar se realizará el procedimiento para un solo MIDlet, aunque se explicarán los paso a seguir para formar una suite de MIDlets.

    Un archivo JAR esta formado por los siguientes elementos:

    a. Un archivo manifiesto que describe el contenido del archivo JAR.
    b. Las clases Java que forman el MIDlet.
    c. Los archivos de recursos usados por el MIDlet.

Creación del archivo manifiesto, este es opcional como se ha mencionado. Este fichero contiene atributos de la forma atributo: valor. Se puede crear desde cualquier editor de texto y tiene la siguiente apariencia:

```
MIDlet-1: Prueba, prueba.png, Prueba
MIDlet-Name: Prueba
        MIDlet-Vendor: Carlos Piñera
                MIDlet-Version: 1.0
                Microedition-Configuration: CLDC-1.0
                        Microedition-Profile: MIDP-1.0
```

La siguiente tabla muestra los atributos que deben formar parte del archivo manifiesto:

Atributo	**Descripción**
MIDlet-Name	Nombre de la MIDlet suite.
MIDlet-Version	Versión de la MIDlet suite.
MIDlet-Vendor	Desarrollador del MIDlet.
MIDlet-n	Contiene una lista con el nombre de la MIDlet suite, icono y nombre del MIDlet en la suite.
Microedition-Configuration	Configuración necesitada para ejecutar el MIDlet.
Microedition-Profile	Perfil necesitado para ejecutar el MIDlet.

Estos son los atributos obligatorios del archivo manifiesto, en la siguiente tabla están los opcionales:

Atributo	Descripción
MIDlet-Description	Descripción de la MIDlet suite.
MIDlet-Icon	Nombre del archivo .png incluido en el JAR.
MIDlet-Info-URL	URL con información sobre el MIDlet.
MIDlet-Data-Size	Número de bytes requeridos por el MIDlet.

En el caso de que se vayan a crear una suite de MIDlets con varios MIDlets habría que definir cada uno de ellos usando el siguiente atributo:

MIDlet-1: Prueba, prueba1.png, Prueba1
MIDlet-2: Prueba, prueba2.png, Prueba2
…

Creación del archivo JAR, para crear este archivo habrá que ir hasta la línea de comandos y escribir lo siguiente:

Jar cmf <archivo manifiesto> <nombrearchivo>.jar –C <clases java> . –C <recursos>

Creación del archivo JAD, este archivo también es opcional pero si es creado deberá poseer una serie de atributos obligatorios. Estos atributos están expuestos en la siguiente tabla:

Atributo	Descripción
MIDlet-Name	Nombre de la MIDlet suite.
MIDlet-Version	Versión de la MIDlet suite.
MIDlet-Vendor	Desarrollador del MIDlet.
MIDlet-Configuration	Configuración necesitada para ejecutar el MIDlet.
MIDlet-Profile	Perfil necesitado para ejecutar el MIDlet.
MIDlet-Jar-URL	URL del archivo JAR de la MIDlet suite.
MIDlet-Jar-Size	Tamaño en bytes del archivo JAR.

Como en el caso del archivo manifiesto, el archivo JAD también puede contener atributos opcionales que son los de la siguiente tabla:

Atributo	Descripción
MIDlet-Data-Size	Mínimo número de bytes de almacenamiento persistente usado por el MIDlet.
MIDlet-Delete-Confirm	Confirmación a la hora de eliminar el MIDlet.
MIDlet-Description	Descripción de la MIDlet suite.
MIDlet-Icon	Archivo .png incluido en el JAR.
MIDlet-Info-URL	URL con información de la MIDlet suite.
MIDlet-Install-Notify	Indica que el AMS (gestor de aplicaciones del dispositivo) indique al usuario la instalación del nuevo MIDlet.

Además de los anteriores, el desarrollador de la MIDlet puede definir atributos adicionales útiles para el MIDlet durante su ejecución.

Las últimas fases que quedan por realizar son las de ejecución y depuración. Estas serán realizadas sobre un emulador, en concreto el Wireless Toolkit que ya esta descargado. Su funcionamiento se explicara en el siguiente apartado, con el resto de aplicaciones y entornos visuales.

3.3.1.2 Desarrollo en entornos visuales (IDE):

Existen numerosas herramientas para crear aplicaciones en java bajo un entorno visual. Estos entornos facilitan mucho el trabajo del programador ya que se puede ir viendo la composición de la aplicación, los errores que va generando, etc... Además suelen incorporar debugadores y emuladores muy útiles a la hora de probar las aplicaciones y de corregir los errores. De entre todas las herramientas existentes en el mercado, y después de efectuar muchas pruebas con ellas en este proyecto se va a utilizar Netbeans IDE ya que aglutina todas las herramientas en un mismo entorno sin necesidad de otros añadidos.

Estas han sido elegidas por su facilidad de uso y su integración ya que contienen todos los elementos necesarios directamente integrados, consiguiendo una comodidad hasta ahora inexistente en el resto de entornos probados. Netbeans es un IDE creado por Sun para programar en Java. Da la posibilidad de crear aplicaciones para todas sus plataformas entre ellas J2ME, ya que desde la misma página de Sun se puede descargar la herramienta con diversos módulos que se integran con la misma en un solo click. Wireless Toolkit también pertenece al conjunto de aplicaciones de Sun y es un emulador muy potente y coherente con el entorno de programación CLDC/MIDP. A pesar de que Netbeans ya incluye un emulador integrado se ha decidido usar esta aplicación ya que es realmente eficiente y muy útil.

Instalación de Netbeans IDE y el modulo Mobility: Directamente desde la página de la herramienta es posible descargarla, este IDE es totalmente gratuito y su página oficial es: http://www.netbeans.org

Es una creación de Sun y hoy por hoy una de sus puntas de lanza ya que pretenden que este IDE acabe sirviendo de base para cualquier programador del mundo que pretenda crear una aplicación en cualquier lenguaje informático existente. En la misma página se encuentran todos los módulos (ad-ons) necesarios. En este caso la dirección web de lo que se necesita y sobre lo que versa la explicación es la siguiente:

http://java.sun.com/javase/downloads/netbeans.html

Este enlace descargará en el ordenador la versión 5.5.1 de Netbeans con el jdk incluido. Desde el siguiente enlace habrá que descargar el Mobility Pack para CLDC y Netbeans 5.5.1. Esta herramienta es indispensable para la creación de aplicaciones en J2ME. Su dirección web es la siguiente:

http://www.netbeans.info/downloads/index.php?p=4

En la página oficial se encuentran otros módulos muy interesantes a los que es recomendable echar un vistazo, multitud de tutoriales, manuales y ejemplos de aplicaciones creadas. Pero que en este caso no serán necesarias.

En el apartado anterior se descargo Wireless Toolkit para CLDC, esta es una herramienta simplemente de emulación que se podría utilizar, pero que no hará falta ya que Netbeans lo incluye en si mismo.

Una vez se han descargado los archivos a nuestro ordenador, se puede proceder a la instalación de los mismos. Para ello primero se deberá instalar Netbeans IDE 5.5.1, simplemente hay que hacer doble click en el archivo descargado e instalarlo en cualquier carpeta (p.e. c:\Netbeans). El programa trae multitud de aplicaciones con el que son necesarias, se deberá ir confirmando la instalación de cada una de ellas y eligiendo los directorios de instalación pertinentes (p.e. instalar el jdk en el caso de que no este ya instalado, es necesario elegir una carpeta como puede ser c:\jdk). Una vez instalado Netbeans y todo lo que trae con el (jdk, pearl, etc…) se procederá a la instalación del Mobility Pack. Su instalación es similar a la de Netbeans, se deberá hacer doble click sobre el mismo y el se encarga de buscar Netbeans e instalarse donde corresponda.

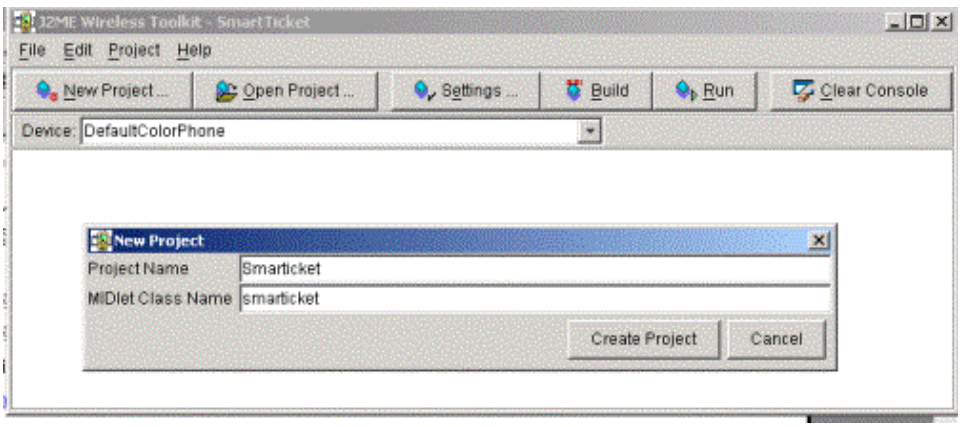


Figura 61. Captura de pantalla de la apariencia del programa Wireless Toolkit de Sun

Desarrollo de aplicaciones en Netbeans IDE: Una vez instalada la herramienta Netbeans podrá dar comienzo el desarrollo de aplicaciones bajo este sistema. Aparecerá un entorno basado en ventanas donde están todas las herramientas necesarias para llevar a buen puerto la creación, diseño y compilación de aplicaciones. La siguiente figura corresponde a Netbeans.

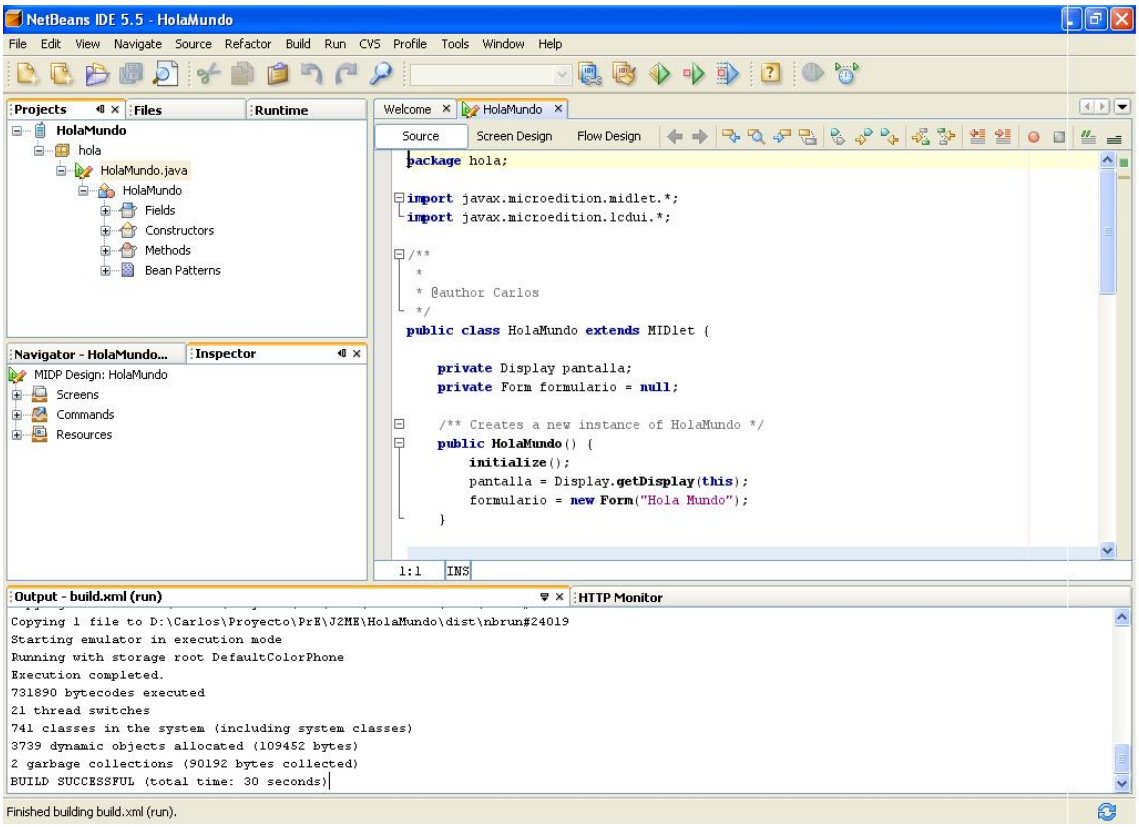
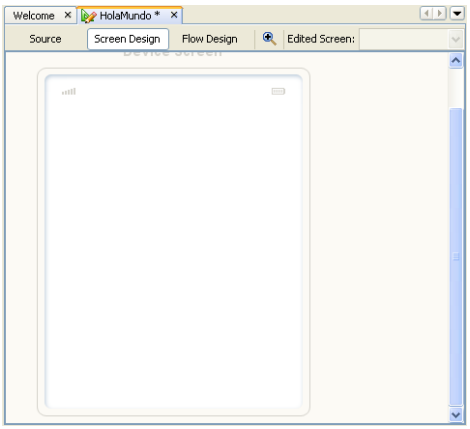
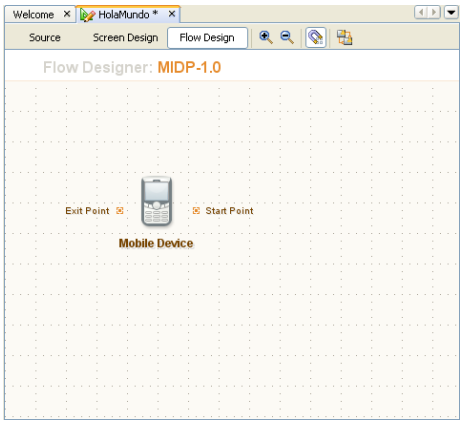


Figura 62. Captura de pantalla de la apariencia del programa Netbeans de Sun.

Como se puede observar, en esta herramienta es posible realizar todas las fases de desarrollo de aplicaciones MIDP.

- En el cuadro superior derecho dispone de un editor de texto totalmente integrado para crear el código fuente (pestaña Source). Además de dos pestañas llamadas Screen Design y Flow Design que sirven para poder diseñar aplicaciones mediante el agregado de elementos directamente sobre la pantalla del terminal y diseño mediante diagramas de flujo respectivamente. Las figuras que vienen a continuación reflejan estas pestañas.





Figuras 63 y 64. Capturas de pantalla de las ventanas Screen Design y Flow Design de Netbeans.

- Una vez creado el código de la aplicación es posible su compilación ya que el entorno trae un compilador (jdk) integrado en el mismo con todas las librerías necesarias. Para compilar cualquier archivo simplemente habrá que pulsar con el botón derecho del ratón sobre el archivo a compilar o proyecto (ya que es posible compilar archivos sueltos o proyectos completos) en la ventana superior izquierda llamada projects y elegir entre las opciones del desplegable la que más convenga. Según el tipo de archivo sobre el que se pinche habrán tres opciones principales (después de poner las opciones se añadirán unas figuras ilustrativas del proceso con cada opción):

    i. Si se pincha sobre un archivo (.java) se deberá elegir la opción Compile File o pulsar F9.

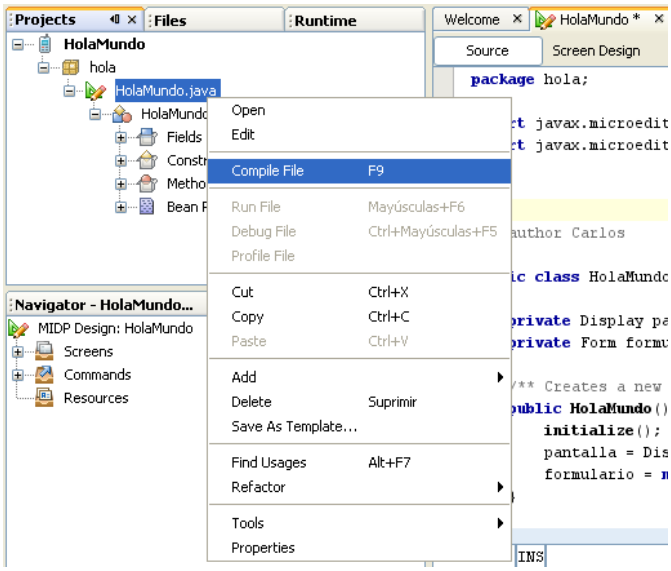


Figura 65. Captura de pantalla de menú Netbeans.

ii. Si se pincha sobre un paquete (package) se deberá elegir la opción Compile Package o pulsar F9.

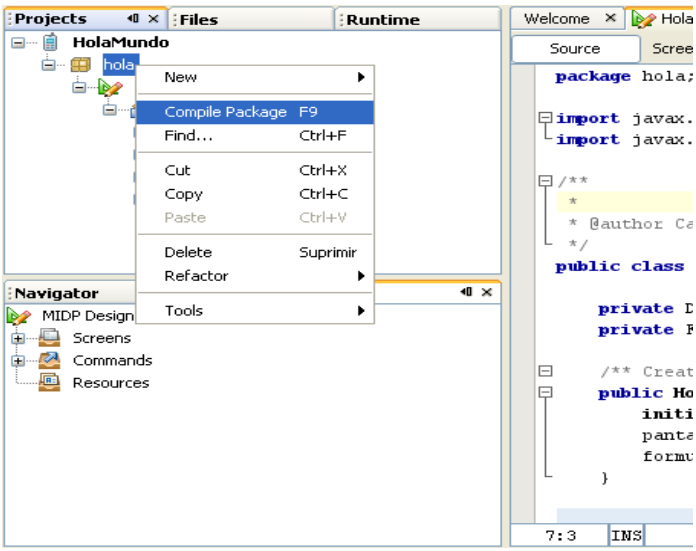


Figura 66. Captura 1 de pantalla de Netbeans

iii. Si se pincha sobre un proyecto (project) se elegirá la opción Build Project.

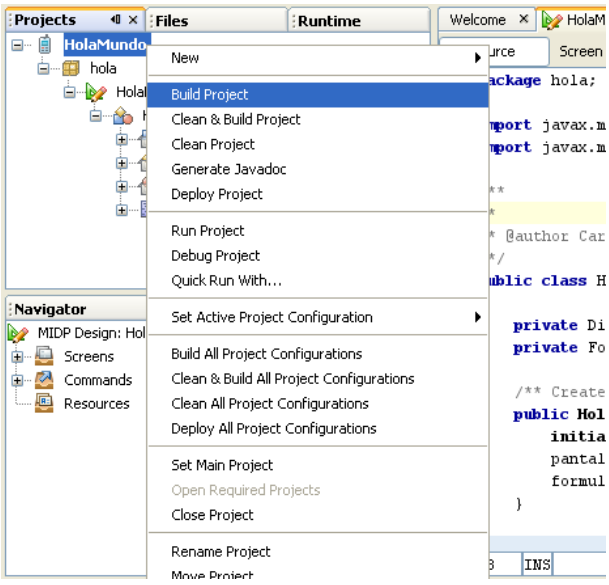


Figura 67. Captura 2 de pantalla de Netbeans

- El proceso de preverificación se realiza automáticamente después de la compilación.
- Una vez que es ejecutada la compilación de un proyecto completo (Build Project) el proceso de empaquetado (generación archivos JAR y JAD) también es automático. Netbeans se encarga de generarlos y guardarlos en un subdirectorio perteneciente al directorio creado para el proyecto llamada dist.
- Las fases de ejecución y depuración también se pueden realizar con esta herramienta, dispone de multitud de opciones y de una barra específica para ello ubicada en la parte superior izquierda (observar figura siguiente). Además de la barra también se podrán utilizar estas herramientas a través de tres pestañas situadas en la parte superior llamadas Run, CVS y Profile. Estas herramientas serán explicadas más adelante con más profundidad.

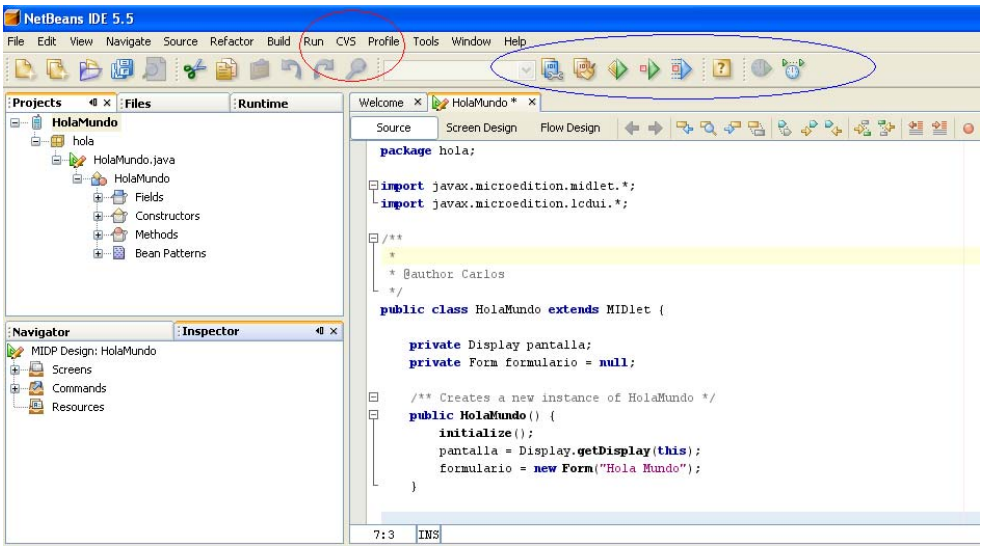


Figura 68. Captura de pantalla de Netbeans con las barras y botones de depuración y ejecución marcados

Una vez explicado un poco por encima el entorno en el que se va a trabajar, ahora se creará un nuevo proyecto desde cero, explicando uno a uno los pasos necesarios para el correcto funcionamiento y creación de todo lo necesario para la instalación de una aplicación en un MID.

Para empezar a utilizar Netbeans crearemos un nuevo proyecto consistente en un Hola Mundo, esta es una aplicación muy sencilla que solo mostrará por pantalla el texto "Hola Mundo" pero que servirá para ir cogiéndole el aire a este nuevo entorno y a las librerías especificas de J2ME, sobre todo a las Microedition.

Lo primero que se deberá hacer es abrir Netbeans, es un entorno que contiene gran cantidad de componentes por lo tanto su carga en pantalla es un poco lenta pero eficiente. Para comenzar a utilizar Netbeans se creará un nuevo proyecto pulsando la pestaña File que se encuentra en la parte superior izquierda del IDE. Dentro de esta se seleccionara la opción New Proyect y a continuación se abrirá una pantalla como la figura que viene a continuación.

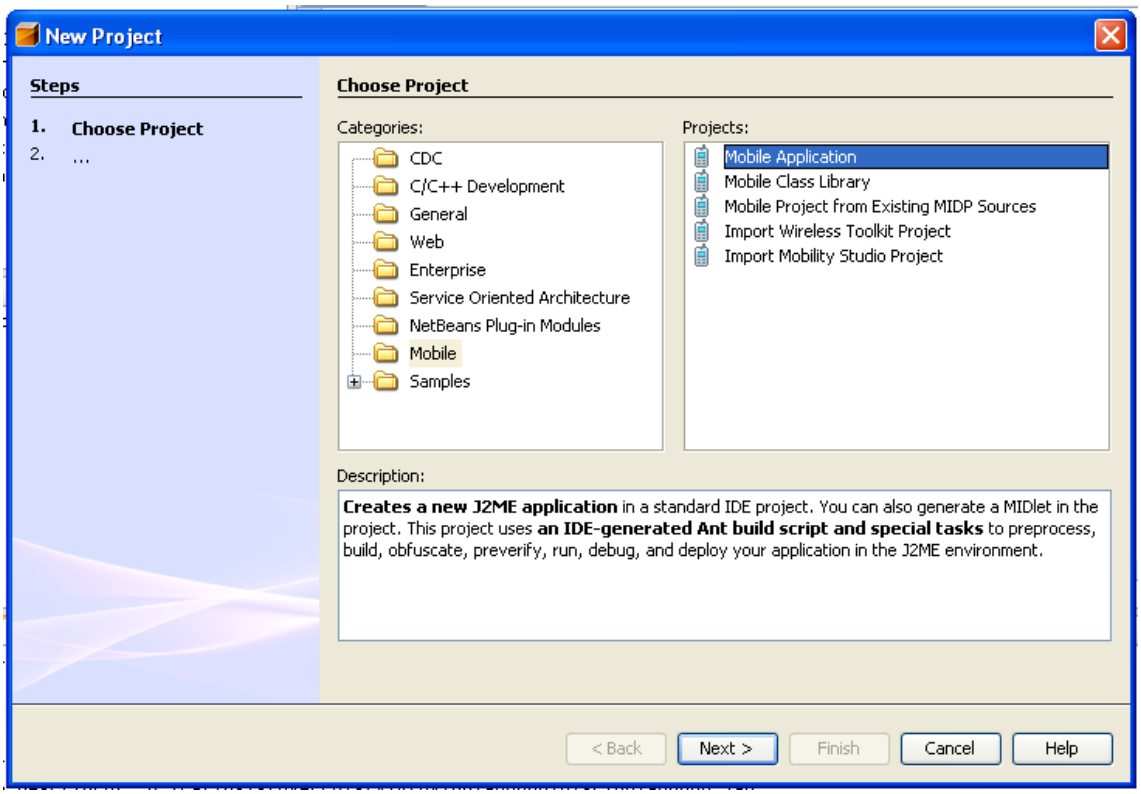


Figura 69. Captura de pantalla de la creación de un proyecto nuevo con Netbeans

En este caso se pueden ver muchas categorías en la parte izquierda del cuadro, esto es debido a que Netbeans soporta gran cantidad de posibles proyectos a ejecutar pero esta parte del proyecto se centrará en la categoría Mobile de la cual, como se puede ver, cuelgan cinco posibles tipos de proyectos cada uno con una descripción del mismo en el cuadro Description de abajo.

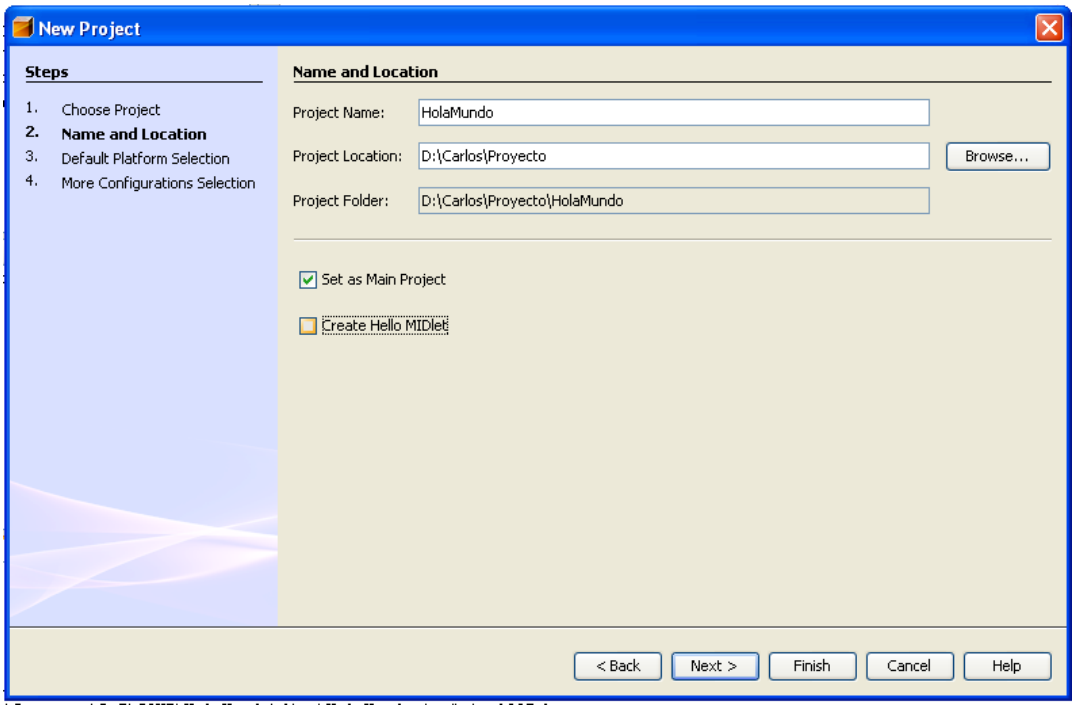


Figura 70. Captura de pantalla de otra ventana de la creación de un nuevo proyecto.

Para esta primera toma de contacto se elegirá Mobile Aplication (lo que se podría llamar directamente MIDlet). Una vez escogida se pinchara en el botón que pone Next para pasar al siguiente cuadro.

En esta figura se puede observar que se ha llegado a la parte de definición del proyecto, se deberá poner un nombre al mismo (en este caso HolaMundo), este nombre de proyecto será el mismo nombre que tendrá la aplicación .java que se va a crear. También se deberá elegir una ubicación para guardar el proyecto. No existen restricciones a la hora de elegir ubicación para los proyectos se podrán poner donde cada uno quiera. Es importante desmarcar la casilla Create Hello MIDlet ya que esta lo que hace es generar un MIDlet con un código de Hola Mundo pero que en este caso no es interesante, ya que ahora se creara el MIDlet especifico con el código que interese. Una vez hecho esto se pulsara Next de nuevo y pasaremos a la siguiente pantalla que pertenece a la elección de plataforma Symbian, perfil y configuración adecuada.

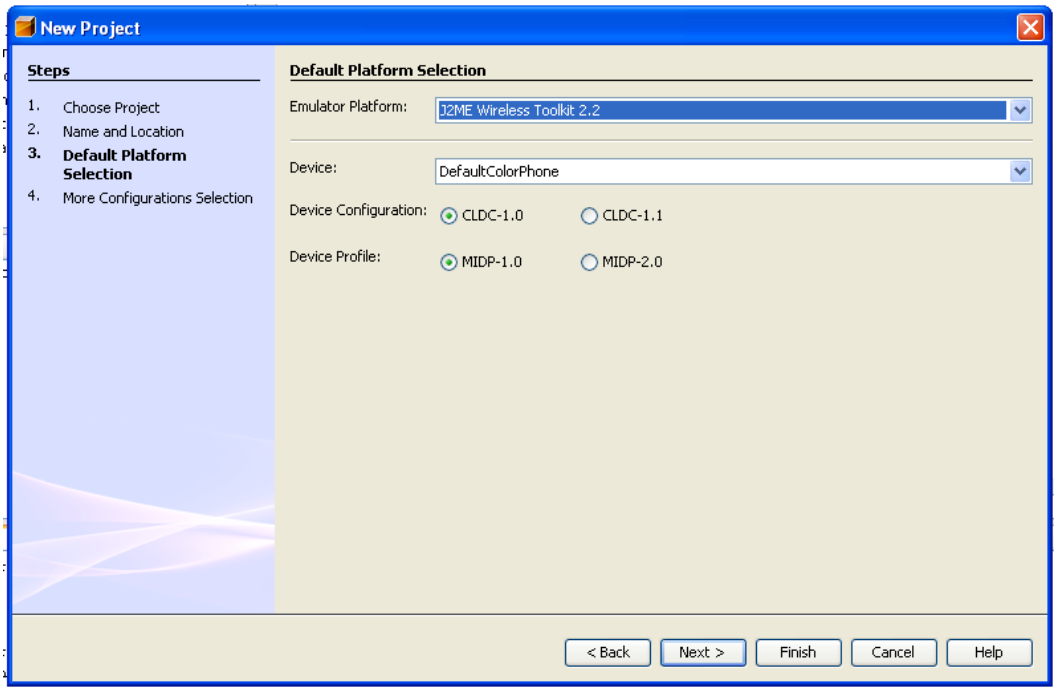


Figura 71. Captura de pantalla de la creación de un nuevo proyecto con Netbeans (Plataforma).

Como se puede observar en esta ventana, Netbeans trae integrado un emulador llamado J2ME Wireless Toolkit 2.2, el cual es suficiente para emular cualquier aplicación que sea creada. De todos modos por si este no fuese suficiente se pueden instalar otros emuladores pertenecientes o no a Sun. Para este proyecto no hará falta ningún otro emulador.

En el desplegable Device se elige el tipo de terminal para el que se esta realizando la aplicación, esto no es muy importante, ya que solo servirá para comprobar que nuestra aplicación encaja bien en el terminal para la que se esta realizando. Hay cuatro posibilidades para el mismo: DefaultColorPhone, DefaultGrayPhone, MediaControlSkin, QwertyDevice. El modelo de terminal con el que han sido probadas todas las aplicaciones

creadas en este proyecto es un Sony Ericsson (P800) y para el mismo siempre se elegirá la opción DefaultColorPhone ya que encaja perfectamente con sus características. En Device Configuration se deberá elegir entre CLDC 1.0 y CLDC 1.1 según las funcionalidades que queramos otorgar a la aplicación, anteriormente ya se explicaron las particularidades de cada una de estas configuraciones. En este caso se elegirá CLDC 1.0 porque no es necesario más. Para terminar se deberá elegir el perfil MIDP mas adecuado, en este caso para esta aplicación tan sencilla que se esta intentando crear con MIDP 1.0 todo funcionara correctamente, mas adelante se utilizará el otro perfil seleccionable, anteriormente también fueron explicadas las diferencias entre ambos perfiles por si es necesario saber que es requerido exactamente.

Una vez seleccionado todo se pasara a la siguiente ventana en la cual se terminará de configurar todo para crear el proyecto.

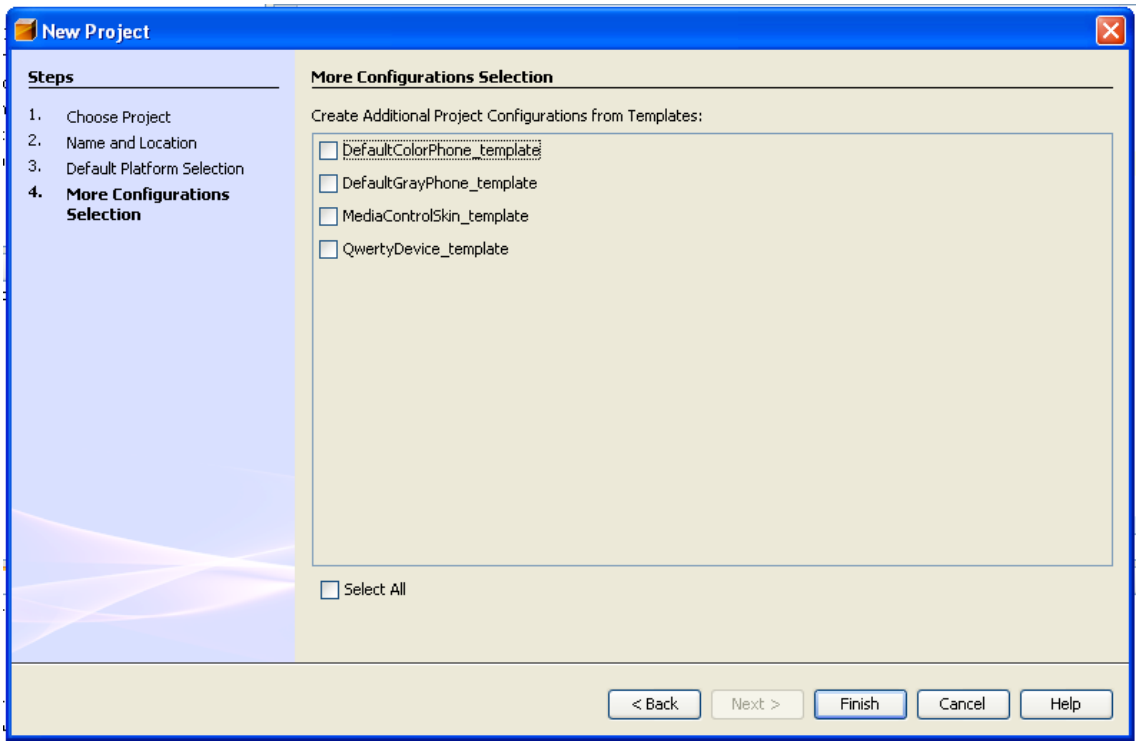


Figura 72. Captura de pantalla de la última ventana de creación de un proyecto nuevo con Netbeans.

Como se puede ver, en esta ventana se podrá seleccionar alguna otra configuración distinta. Esto sirve para crear la misma aplicación sobre diferentes configuraciones y perfiles, pero en este caso no se hará uso de esto, ya que el dispositivo del que se dispone en este proyecto es único y no se podrían probar otras configuraciones distintas a las que encajan con el MID utilizado. Una vez terminada la configuración del proyecto se pulsara el botón Finish y se creara el mismo.

El proyecto creado estará totalmente vacio, aparecerá en la ventana Proyects en la parte superior izquierda. Una vez realizados todos estos pasos todo esta preparado para empezar a codificar el código de la aplicación.

3.3.1.3 Creación y codificación de la aplicación Hola Mundo

Es importante en java modularizar todos los proyectos y una buena técnica es crear paquetes que contengan códigos específicos para cada tipo de aplicación, por ello, al principio de crear un nuevo proyecto se creará dentro del mismo un paquete que contendrá todas las clases java del proyecto. Esto permitirá mover todas las clases en un solo paso sin perder nunca ninguna clase necesaria para el funcionamiento de la aplicación. Por tanto lo primero que se hará será pulsar con el botón derecho sobre el proyecto recién creado y elegir la opción New -> Java Package y ponerle un nombre especifico al paquete donde pondremos todos nuestros códigos fuente. Una vez hecho esto, dentro del paquete recién creado volveremos a pinchar con el botón derecho y en este caso se elegirá la opción New -> MIDlet. Esto generará una nueva clase con el nombre que elijamos que contendrá las librerías necesarias y los métodos básicos a implementar para el funcionamiento del MIDlet. La clase generada será algo así:

```
package holamundo;

import javax.microedition.midlet.*;

public class HolaMundo extends MIDlet {

    public HolaMundo(){

//Este es el constructor de la clase donde inicializaremos las variables.

    }

    public void startApp() {

//Aquí se incluiría el código que queremos que el MIDlet ejecute cuando
se active.

    }

    public void pauseApp() {

//En este método incluiremos el código que queremos que se ejecute cuando
//el Midlet pase al estado de pausa. Implementar este método es opcional.

    }

      public void destroyApp(boolean unconditional) {

//Aquí va el código del estado Destruido. Normalmente aquí se liberan los
//recursos.

    }

}
```

Como se puede observar la clase heredara de MIDlet y esta compuesta de tres métodos básicos que deberemos implementar: startApp, pauseApp y destroyApp. Estos métodos son necesarios ya los posibles estados de un MIDlet son tres:

- Activo: El MIDlet esta actualmente en ejecución.
- Pausado: El MIDlet no esta en ejecución. En este estado el MIDlet no debe usar ningún recurso compartido. Para volver a pasar a ejecución tiene que cambiar su estado a activo.
- Destruido: El MIDlet no esta en ejecución ni puede transitar a otro estado. Además se liberan todos los recursos ocupados por el MIDlet.

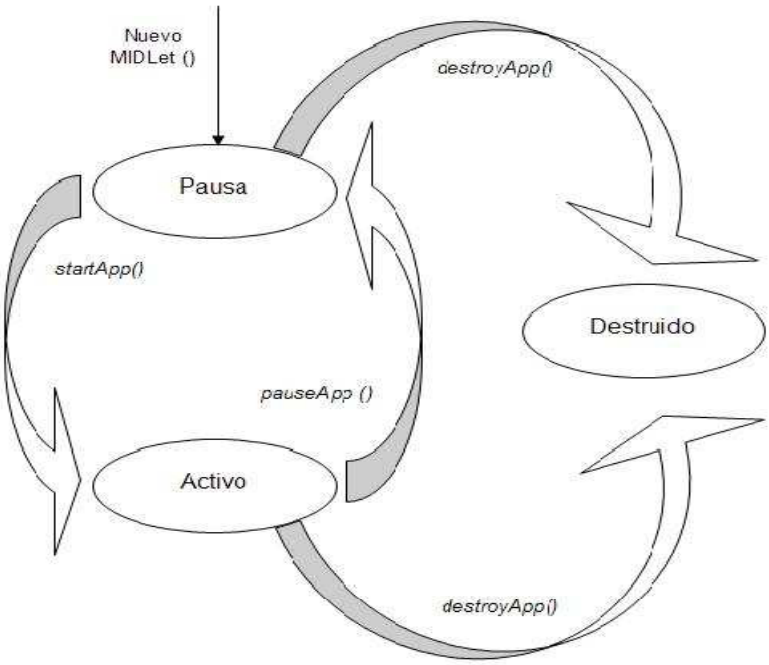


Figura 73. Diagrama de estados de un MIDlet en ejecución

Como se puede observar en el diagrama, un MIDlet puede cambiar de estado mediante una llamada a los métodos antes mencionados y de obligatoria implementación. El gestor de aplicaciones cambia el estado de los MIDlets haciendo una llamada a cualquiera de estos métodos. Un MIDlet también puede cambiar de estado por si mismo. En la figura se puede observar que cuando se realiza la llamada al constructor de MIDlets este crea el MIDlet y lo inicializa en el estado "Pausa", una vez en este estado el MIDlet podría pasar en cualquier momento a un estado "Activo" o directamente a estado "Destruido". En el estado "Activo" el MIDlet ocupara los recursos y realizara la función para la que fue diseñado pudiendo después volver a "Pausa" (por terminar su cometido, o por orden explicita del usuario) o a estado "Destruido". En ambos devolverá todos los recursos y realizará el cometido específico de ese estado.

Una vez ya se conocen los métodos básicos de un MIDlet y sus estados se va a examinar en profundidad el paquete javax.microedition.midlet el cual será necesario conocer para la creación de la aplicación Hola Mundo. En este punto es mejor conocer bien los paquetes de los que se dispone antes de empezar a codificar a lo loco sin la información necesaria.

El paquete javax.microedition.midlet define las aplicaciones MIDP y su comportamiento con respecto al entorno de ejecución. Una aplicación creada usando MIDP es un *MIDlet*. En la siguiente tabla se puede ver cuáles son las clases que están incluidas en este paquete:

Clases	**Descripción**
MIDlet	Aplicación MIDP.
*MIDlet*stateChangeException	Indica que el cambio de estado ha fallado.

Ahora se verán en profundidad cada una de estas clases con sus correspondientes métodos.

Clase MIDlet

public abstract class MIDlet

Un MIDlet es una aplicación realizada usando el perfil MIDP. La aplicación debe extender a esta clase para que el AMS pueda gestionar sus estados y tener acceso a sus propiedades. El MIDlet puede por sí mismo realizar cambios de estado invocando a los métodos apropiados. Los métodos de los que dispone esta clase son los siguientes:

- *protected MIDlet()*

  Constructor de clase sin argumentos. Si la llamada a este constructor falla, se lanzaría la excepción SecurityException.

- *public final int checkPermission(String permiso)*

  Consigue el estado del permiso especificado. Este permiso está descrito en el atributo MIDlet-Permission del archivo JAD. En caso de no existir el permiso por el que se pregunta, el método devolverá un 0. En caso de no conocer el estado del permiso en ese momento debido a que sea necesaria alguna acción por parte del usuario, el método devolverá un -1. Los valores devueltos por el método se corresponden con la siguiente descripción:
  - 0 si el permiso es denegado
  - 1 si el permiso es permitido
  - -1 si el estado es desconocido

- *protected abstract void destroyApp(boolean incondicional) throws MIDletstateChangeException*

  Indica la terminación del *MIDlet* y su paso al estado de "Destruido". En el estado de "Destruido" el *MIDlet* debe liberar todos los recursos y salvar cualquier dato en el almacenamiento persistente que deba ser guardado. Este método puede ser llamado desde los estados "Pausa" o "Activo". Si el parámetro 'incondicional' es *false*, el MIDlet puede lanzar la excepción MIDletstateChangeException para indicar que no puede ser destruido en este momento. Si es *true*, el *MIDlet* asume su estado de destruido independientemente de como finalice el método.

- *public final String getAppProperty(String key)*

  Este método proporciona al *MIDlet* un mecanismo que le permite recuperar el valor de las propiedades desde el AMS. Las propiedades se consiguen por medio de los archivos *manifest* y *JAD*. El nombre de la propiedad a recuperar debe ir indicado en el parámetro key. El método nos devuelve un String con el valor de la propiedad o *null* si no existe ningún valor asociado al parámetro key. Si key es null se lanzará la excepción NullPointerException.

- *public final void notifyDestroyed()*

  Este método es utilizado por un MIDlet para indicar al AMS que ha entrado en el estado de "Destruido". En este caso, todos los recursos ocupados por el MIDlet deben ser liberados por éste de la misma forma que si se hubiera llamado al método MIDlet.destroyApp(). El AMS considerará que todos los recursos que ocupaba el MIDlet están libres para su uso.

- *public final void notifyPaused()*

  Se notifica al AMS que el MIDlet no quiere estar "Activo" y que ha entrado en el estado de "Pausa". Este método sólo debe ser invocado cuándo el MIDlet esté en el estado "Activo". Una vez invocado este método, el MIDlet puede volver al estado "Activo" llamando al método MIDlet.startApp(), o ser destruido llamando al método MIDlet.destroyApp(). Si la aplicación es pausada por sí misma, es necesario llamar al método MIDlet.resumeRequest() para volver al estado "Activo".

- *protected abstract void pauseApp()*

  Indica al MIDlet que entre en el estado de "Pausa". Este método sólo debe ser llamado cuándo el MIDlet esté en estado "Activo". Si ocurre una excepción RuntimeException durante la llamada a MIDlet.pauseApp(), el MIDlet será destruido inmediatamente. Se llamará a su método MIDlet.destroyApp() para liberar los recursos ocupados.

- *public final boolean platformRequest(String url)*

  Establece una conexión entre el MIDlet y la dirección URL. Dependiendo del contenido de la URL, el dispositivo ejecutará una determinada aplicación que sea capaz de leer el contenido y dejar al usuario que interactúe con él. Si, por ejemplo, la URL hace referencia a un archivo JAD o JAR, el dispositivo entenderá que se desea instalar la aplicación asociada a este archivo y comenzará el proceso de descarga. Si, por el contrario, la URL tiene el formato tel:<número>, el dispositivo entenderá que se desea realizar una llamada telefónica.

- *public final void resumeRequest()*

  Este método proporciona un mecanismo a los *MIDlets* mediante el cual pueden indicar al AMS su interés en pasar al estado de "Activo". El AMS, en consecuencia, es el encargado de determinar qué aplicaciones han de pasar a este estado llamando al método MIDlet.startApp().

- *protected abstract void startApp() throws MIDletstateChangeException*

  Este método indica al MIDlet que ha entrado en el estado "Activo". Este método sólo puede ser invocado cuándo el MIDlet está en el estado de "Pausa". En el caso de que el MIDlet no pueda pasar al estado "Activo" en este momento pero si pueda hacerlo en un momento posterior, se lanzaría la excepción MIDletstateChangeException.

Clase MIDletChangeStateException

public class MIDletstateChangeException extends Exception

Esta excepción es lanzada cuando ocurre un fallo en el cambio de estado de un MIDlet.En la siguiente tabla se puede ver un resumen de los métodos que dispone la clase.

Constructores	
protected	MIDlet()
Métodos	
int	checkPermission (String permiso)
protected abstract void	destroyApp (boolean unconditional)
String	getAppProperty(String key)
void	notifyDestroyed()
void	notifyPaused()
protected abstract void	pauseApp()
boolean	platformRequest()
void	resumeRequest()
protected abstract void	startApp()

Una vez conocido el paquete javax.microedition.midlet se puede intentar el asalto a la codificación de la aplicación anteriormente planteada, Hola Mundo. Para ello se implementarán cada uno de los métodos necesarios en la misma, explicando después su compilación, uso del emulador y uso del debugador incluidos en Netbeans.

El código de la aplicación será el siguiente:

```
package holamundo;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


public class HolaMundo extends MIDlet {
private Display pantalla;                        //Creo un objeto Display
private Form formulario = null;                  //Creo un objeto
formulario y lo inicializo a null

     public HolaMundo(){
      pantalla = Display.getDisplay(this);    //Añado pantalla a el
                                                     //escuchador

      formulario = new Form("Hola Mundo");      //Relleno formulario
     }

     public void startApp() {
      pantalla.setCurrent(formulario);    //Selecciono pantalla
         }

     public void pauseApp() {}

     public void destroyApp(boolean unconditional) {
      pantalla = null;                             //Elimino pantalla
      formulario = null;                           //Elimino formulario

     notifyDestroyed();             //Le notifico al AMS que el MIDlet ha
                                    //sido destruido para que libere los
                                    //recursos consumidos.
         }
     }
```

Como se puede comprobar la dificultad del código es mínima, pero sirve como una buena iniciación a la programación con J2ME ya que se han visto las clases mínimas que necesitas implementarse en un MIDlet. Al ser un MIDlet se puede observar que la clase hereda directamente de MIDlet, que se ha sobrescrito el constructor (HolaMundo) y que en este caso no interesaba que el MIDlet pasara por el estado de "Pausa", por ello la implementación de el método pauseApp no es necesaria, pero si obligatorio poner la cabecera por heredar de MIDlet que posee a la misma entre sus tres métodos abstractos.

El siguiente paso es compilar el programa para asegurar que no contiene fallos, para ello es necesario pinchar sobre el paquete generado anteriormente (llamado holamundo). En este caso se podría compilar solo el archivo HolaMundo.java ya que el proyecto solo contiene este archivo, pero cuando la dificultad aumenta suele haber mas de una clase que normalmente van interrelacionadas entre ellas. Por ello lo mas normal es compilar todo el paquete para que Netbeans pueda realizar sus asignaciones y no genere errores por no encontrar alguna Clase. Una vez compilado el paquete compilaremos también el proyecto para que toda la implementación se guarde correctamente en el disco duro.

3.3.1.4 Manejo del Emulador de Netbeans (Wireless Toolkit)

El manejo del emulador en Netbeans es muy sencillo. Una vez tenemos el código compilado y sin errores, este puede ser ejecutado. Para ejecutar el emulador sobre una aplicación se deberá tener el proyecto con la aplicación en pantalla, una vez en este punto, hay dos maneras de lanzar el emulador:

- La primera de ellas es con la barra de emulación y debugación que se encuentra en la parte superior izquierda, justo encima de la ventana donde se genera el código. Esta barra contiene 8 botones que serán explicados a continuación.
- La segunda es con la pestaña desplegable de la parte superior del programa llamada Run. Este desplegable también contiene varias opciones que serán comentadas a continuación.

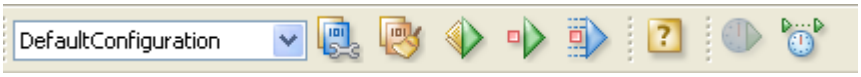


Figura 74. Barra de emulación y debugación de Netbeans

Esta figura corresponde a la barra de emulación y debugación que había sido nombrada anteriormente. Primero se explicaran los botones de emulación ya que los de debugación serán vistos en el siguiente apartado llamado Manejo del debugador de Netbeans. Los botones correspondientes al emulador son:

- El primer desplegable se corresponde con la configuración que fue elegida en la creación del proyecto (como se puede observar es DefaultConfiguration).
- El primer y el segundo botón son para compilar el proyecto directamente (build main project) y para compilar y limpiar el proyecto (clean and build main proyect) respectivamente.
- El tercer botón es el más importante en este caso, este es el botón de emulación simple. Como antes ya se ha mencionado una vez tienes el proyecto compilado, pulsando este botón se abrirá la pantalla de emulación para poder ver la aplicación creada ejecutándose. Es así de sencillo, se pulsa el botón y al momento se inicia el emulador. Con el emulador se puede interactuar, o sea, se puede picar en las teclas del dispositivo que aparece y este reaccionara de una manera u otra según la tecla que sea clicada. Es muy realista, permite la perfecta comprobación del funcionamiento de una aplicación sin el embrollo de instalar la aplicación en el MID y disminuyendo los riesgos de la posible instalación en el MID de una aplicación inestable. En la figura de al lado se muestra una vista del emulador con la aplicación Hola Mundo ejecutándose en el.

Figura 75. Captura de pantalla del emulador de Netbeans

- El cuarto botón es para lanzar el debugador y el emulador a la vez. Este botón (Debug Main Proyect) lanzará la KVM (Kilobyte Virtual Machine) ejecutando emulador y debugador al unísono, permitiendo al programador comprobar los fallos del programa directamente sobre el emulador. Utilizando esta herramienta se puede interactuar con el emulador y el debugador ira resaltando por pantalla cada una de las incidencias o acciones que están ocurriendo.
- El quinto botón se verá en el siguiente apartado ya que es perteneciente al debugador puramente, sirve para ejecutar una aplicación configurando una conexión. Con el se podrán probar aplicaciones que requieren uso de sockets.
- El sexto botón consiste en una ayuda en tiempo de ejecución, con el se podrán consultar detalles de la implementación de la aplicación dinámicamente.
- El séptimo y octavo botón sirven para comprobar el resto de detalles de la aplicación como el consumo de recursos de la misma, la velocidad de ejecución, el modo en que se ejecuta un trozo de código especifico. Son meramente para optimización de los recursos e información sobre la aplicación.

La segunda manera de lanzar el emulador antes mencionada es con el desplegable de la parte superior de la pantalla llamado Run. Este incorpora algunas opciones más que la barra hasta ahora explicada y su uso es similar. Contiene los botones de la barra, además de algunos otros para ejecución paso a paso, introducción de Breakpoints, control de ejecución, saltos en código, etc.

3.3.1.5 Manejo del debugador de Netbeans

Esta herramienta incorporada a Netbeans es una maravilla. Permite al programador comprobar los fallos en sus aplicaciones J2ME sin las complicaciones normales de debugación en Java. Gracias a esta herramienta ya no es necesario el uso de los famosos system.out.println() tan usados en Java para depurar los programas.

En el apartado anterior ya se vieron los puntos desde los que acceder a esta herramienta. Una vez es pulsado el botón de lanzamiento del debugador se abre en la parte inferior de la pantalla una nueva pestaña junto a la consola de compilación (llamada build), esta tiene como nombre Debugger console y en ella serán mostradas todas las incidencias producidas durante la ejecución de la aplicación.

A el debugador se le pueden aplicar todos los métodos conocidos de debugación de otros lenguajes, como introducción de breakpoints, ejecución paso a paso, ejecución de trozos de código, etc… Su funcionamiento es muy sencillo y basta con pulsar en el desplegable Run o en el botón especifico de debugación para comprobarlo. Debajo se muestra una figura con la pestaña que se crea al lanzar el debugador.

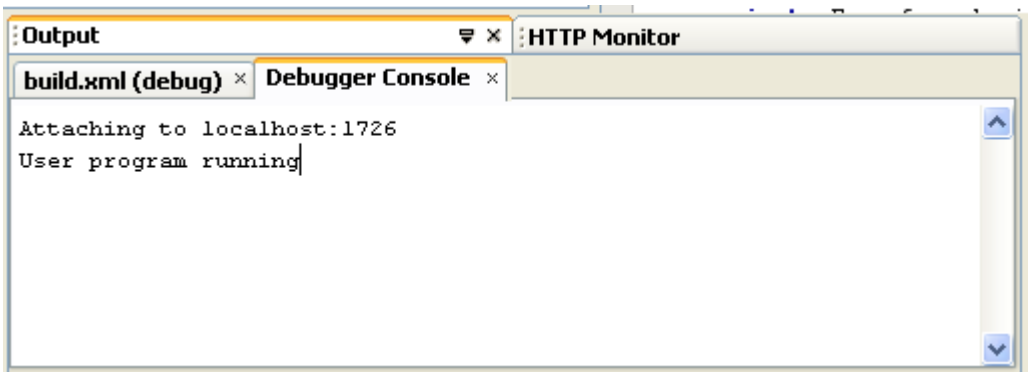


Figura 76. Captura de pantalla de la pestaña de debugación

3.3.1.6 Ventajas de uso de Netbeans IDE para el desarrollo de aplicaciones en J2ME

Netbeans es un IDE muy completo que elimina mucha de la complejidad que requiere la programación para una plataforma hasta ahora desconocida como es J2ME. Aquí están sus principales ventajas:

- *Mejorar la conexión cliente J2ME hacia web.* Gran facilidad para acceder a servicios web, y a otros datos del servidor desde los MIDlets vía Servlets. Incluye soporte para JSR-172

- *Soluciona en gran parte el problema de la fragmentación.* Uno de los aspectos más peliagudos del desarrollo de aplicaciones para J2ME es el conocido como fragmentación del mercado. Escribir una aplicación que va a correr en dispositivos diversos puede llegar a ser un reto hercúleo. Diferencias en los aspectos físicos del dispositivo, como puede ser el tamaño de la pantalla, así como diferencia en los cuestiones del software soportado por cada dispositivo, puede hacer que una aplicación necesite variaciones, tanto en código como en configuración.

  Netbeans hace frente a todos estos retos usando el concepto de configuración, lo que permite desarrolla tu aplicación para múltiples dispositivos añadiendo y ejecutando código especifico de un dispositivo como una configuración dentro de la misma aplicación. Se deberá crear una configuración por cada jar que se pretenda distribuir. Por ejemplo, si tu tienes en mente soportar tres tipos distintos de tamaño de pantalla usando dos api especificas de un vendedor, habrá que crear 6 jar distintos. Pero esto solo sería necesario si quisiésemos sacar partido de esas características especiales de unos dispositivos específicos.

  Es posible desarrollar la aplicación siguiendo el mínimo común denominador, la especificación estándar de J2ME y conseguir que la aplicación se vea en todos los móviles que queramos sin meter código particular.

- *Facilidades de distribución.* Con Netbeans es posibles distribuir tus paquetes de MIDlets directamente usando los protocolos WebDAV, FTP o SCP.

- *Proyectos basados en ANT*. Al igual que pasaba con el IDE, en Netbeans Mobility pack todos los proyectos están basados en ANT, lo que hace que sea muy fácil adaptarlo a nuestra necesidades. Incluso podemos desarrollar el proyecto fuera de este entorno para posteriormente ejecutarlo dentro Netbeans.

- *Integración con el J2ME Wireless Toolkit 2.2*. El componente Netbeans Mobility Pack proporciona soporte para el desarrollo de características avanzadas de la plataforma. Como firmado de MIDlets, gestión de certificados, mensajería, etc.

- *Soporte para localizar los contenidos del MIDlet.* Nos da la posibilidad de desarrollar la aplicación con capacidad de adaptar los recursos a la zona donde se piense usar el programa.

- *Soporte para los estándares J2ME MIDP 2.0 y CLDC 1.1.* Nos permite trabajar con los dispositivos que se hayan adherido a los últimos estándares de J2ME.

- *Capacidad para añadir emuladores adicionales.* Con Netbeans es posible extender los dispositivos de prueba que ofrece el emulador y añadir los que nos proporcionan los distintos fabricantes de móviles, Nokia, Motorola, Benq, Samsung, Ericsson. Los modernos emuladores desarrollados por estos fabricantes se adaptan al estándar UEI (United Emulator Interface), Netbeans es capaz de detectar estos emuladores y tratar con ellos como si fuesen una parte más del producto.

- *Over-the-Air (OTA) pruebas de descargas*. El emulador simula el comportamiento de los dispositivos móviles reales.

- *Capacidad de ofuscar el código, optimizar el tamaño del código*. Hasta 9 niveles distintos de ofuscación de código lo que nos permite añadir seguridad, mejorar el rendimiento, y optimizar el tamaño de nuestra aplicación.

- *Soporte para Java ME WebServices (JSR 172)*. Escribir aplicaciones que accedan directamente a los servicios web desde el teléfono móvil.

- *Componentes propios.* Netbeans añade algunos componentes visuales a los que ya se ofrecen con la plataforma MIDlet 2.0, como por ejemplo pantallas de espera o tablas, poniendo a nuestra disposición una mayor riqueza visual a la hora de crear aplicaciones j2ME.

- *Preprocesador mejorado para para la fragmentación*. Desarrollar código para múltiples dispositivos con una única línea de código. Ahora se pueden importar proyectos de J2ME Polish sin perder código de preprocesado.

- *Editar el código generado.* Ahora es posible modificar el código generado por el editor visual, por si hiciera falta modificar el comportamiento o los valores de las variables de la aplicación entre el paso de dos pantallas.

- *Depurar la aplicación.* Al estar ejecutándose las aplicaciones de móviles dentro del entorno Netbeans, podemos usar toda la potencia del IDE, y todos sus componentes para que nos ayuden en el desarrollo de las aplicaciones J2ME.

- *Componentes opcionales del J2ME.* No solo es posible desarrollar con las especificaciones estándar del J2ME, también Netbeans no da la posibilidad de hacerlo con los componentes opcionales Bluetooth, java 3D, etc… por si el dispositivo para el que queremos desarrollar tiene esas capacidades.

- *Soporte para CDC.* Recientemente se ha añadido soporte a Netbeans para el desarrollo de aplicaciones que sigan la especificación CDC. Para aquellos dispositivos que cumplan con este estándar, es ahora posible diseñar sus pantallas usando el popular y exitoso Matisse. Y ejecutar el test control de nuestros proyectos usando el JUnit.

Figura 77. Captura de pantalla de la ventana de inicio de Netbeans versión 5.5

3.3.2 Entrada/Salida de datos en J2ME

3.3.2.1 Introducción

En este punto se abordará la entrada y salida de datos en J2ME. Para ello se realizará una aplicación en modo agenda, en la cual se podrán introducir, borrar y modificar datos de una lista. En este punto también se verá en profundidad el paquete RMS (Record Management System o Sistema de gestión de Registros) perteneciente a las librerías J2ME y todas sus clases para poder trabajar con comodidad en este tipo de aplicaciones. Este paquete es utilizado para poder guardar datos en una memoria de tipo no volátil y no perderlos tras la ejecución de la aplicación. La información será guardada en el dispositivo, en una zona restringida para este propósito, la cantidad de memoria existente y la zona reservada para ello dependerá de cada dispositivo.

3.3.2.2 RMS (Record Management System)

Un dispositivo móvil (al menos por ahora) no dispone de disco duro donde almacenar información permanentemente. J2ME resuelve el problema mediante el RMS (Record Management System). RMS es un pequeño sistema de bases de datos muy sencillo, pero que permite añadir información en una memoria no volátil del móvil. RMS no tiene nada que ver con JDBC debido a las limitaciones de los dispositivos J2ME, por lo tanto, el acceso y almacenamiento de la información se hace a mucho más bajo nivel. RMS no puede ser consultado con sentencias SQL ni nada parecido. En una base de datos RMS, el elemento básico es el registro (record). Un registro es la unidad de información más pequeña que puede ser almacenada. Los registros son almacenados en un recordStore que puede visualizarse como una colección de registros. Cuando almacenamos un registro en el recordStore, a éste se le asigna un identificador único que identifica unívocamente al registro.

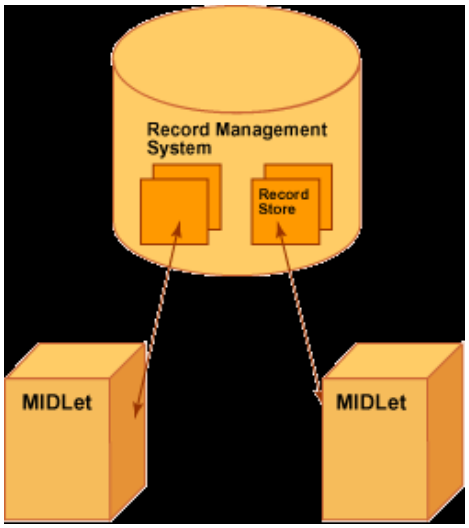


Figura 78. Diagrama de funcionamiento de RMS

El mecanismo básico de almacenamiento de RMS es denominado record store. Un record store es un conjunto de registros, y un registro es un byte array de datos de tamaño variable. Un record store está representado por un objeto de la clase RecordStore. Existen reglas importantes sobre los record store:

- El nombre de un record store consiste en una combinación de hasta 32 caracteres (sensible a las mayúsculas).
- Los record stores son creados por MIDlets. Los MIDlets de un mismo MIDlet suite están almacenados en el mismo espacio de nombres, y por lo tanto, pueden compartir y ver sus contenidos.
- Los record stores creados por MIDlets en un MIDlet suite, no son accesibles para los MIDlets de otros MIDlets suite.
- El nombre de un record store debe ser único en un MIDlet suite.

Un *Record Store* tal como su nombre indica es un almacén de registros. Estos registros son la unidad básica de información que utiliza la clase RecordStore para almacenar datos.

Cada uno de estos registros está formado por dos unidades:

- Un número identificador de registro (*Record* ID) que es un valor entero que realiza la función de clave primaria en la base de datos.
- Un *array* de bytes que es utilizado para almacenar la información deseada.

En la siguiente figura se muestra la estructura de un Record Store:

Record ID	**Datos**
1	Byte [] arrayDatos
2	Byte [] arrayDatos
…	…

Además de un nombre, cada *Record Store* también posee otros dos atributos:

- Número de versión: Es un valor entero que se actualiza conforme vayamos insertando, modificando o borrando registros en el *Record Store*. Podemos consultar este valor invocando al método RecordStore.getVersion().
- Marca temporal: Es un entero de tipo long que representa el número de milisegundos desde el 1 de enero de 1970 hasta el momento de realizar la última modificación en el *Record Store*. Este valor lo podemos obtener invocando al método RecordStore.getLastModified().

Así la estructura de un Record Store se corresponde más con la siguiente figura.

Operaciones con un Record Store

La clase RecordStore proporciona los siguientes métodos para crear, abrir, cerrar y borrar un record store:

- Para crear/abrir un objeto RecordStore se utiliza el método

  public static RecordStore openRecordStore(String recordName, boolean createIfNecessary)

  - Para crearlo se puede utilizar: *RecordStore.openRecordStore (recordStoreName, true)*, de tal forma que si el record store con nombre recordStoreName no existe, se crea. Si existe, se "abre" el existente para trabajar sobre él.

  - Para abrirlo también se puede utilizar: *RecordStore.openRecordStore(recordStoreName, false)*, de tal forma que si el record store no existe, se produce una excepción RecordStoreNotFoundException

- Para cerrar un objeto RecordStore se utiliza el método *public void closeRecordStore ()*. Después de utilizar un record store siempre debe cerrarse.

- Para borrar un objeto RecordStore se utiliza el método *public static void deleteRecordStore (String recordStoreName)*. Antes de borrar un record store es necesario cerrarlo.

Cada record store mantiene al menos un campo de cabecera. Si no existe espacio suficiente para almacenar la cabecera, el record store no se creará y se producirá una RecordStoreFullException. Si ocurre otro tipo de problema, como que el nombre del record store es demasiado largo, o el record store está corrupto, se produce una RecordStoreException.

En la siguiente tabla se pueden ver algunos de los métodos que proporcionan operaciones con los Record Stores:

Métodos	Descripción
String getName()	Devuelve el nombre del Record Store
int getVersion()	Devuelve la versión del Record Store.
long getLastModified()	Devuelve la marca temporal.
int getNumRecords()	Devuelve el número de registros.
int getSize()	Devuelve el número de bytes ocupado por el Record Store.
int getSizeAvailable()	Devuelve el tamaño disponible para añadir registros.
String[] listRecordStores()	Devuelve una lista con los nombres de los Record Stores que existen en la MIDlet suite.
void deleteRecordStore(String name)	Elimina del dispositivo al Record Store especificado por el parámetro 'name'.
RecordEnumeration enumerateRecords(RecordFilter filter, RecordComparator comparator, Boolean actualizado)	Nos devuelve un objeto RecordEnumeration.
void addRecordListener(RecordListener listener)	Añade un 'listener' para detectar cambios en el Record Store.
void removeRecordListener (RecordListener listener)	Elimina un 'listener'.

Una vez creado o abierto un Record Store se puede leer, escribir, modificar o borrar registros según se necesite. Para este fin se puede hacer uso de los métodos de la clase Record Store específicos para ello. En la siguiente tabla se expondrán todos los métodos adecuados para la realización de estas tareas sobre ellos.

Método	Descripción
int addRecord(byte[] datos, int offset, int numBytes)	Añade un registro al Record Store
void deleteRecord(int id)	Borra el registro 'id' del Record Store
Int getNextRecordId()	Devuelve el siguiente 'id' del registro que se vaya a insertar
byte[] getRecord(int id)	Devuelve el registro con identificador 'id'
int getRecord(int id, byte[] buffer, int offset)	Devuelve el registro con identificador 'id' en 'buffer' a partir de 'offset'
Int getRecordSize(int id)	Devuelve el tamaño del registro 'id'
void setRecord(int id, byte[] datonuevo, int offset, int tamaño)	Sustituye el registro 'id' con el valor de 'datonuevo'

De entre todas las operaciones posibles a realizar las más utilizadas suelen ser las de lectura y escritura ya que esta clase esta diseñada para ello. Antes de empezar con el ejemplo propuesto en este apartado se dará una breve explicación de otro tipo de operaciones existentes en este paquete. Estas son del tipo avanzado, en un Record Store además de lo visto también se podrá navegar por el, buscar un registro concreto, determinar un orden específico para los registros o manejar los eventos salientes de los mismos, estas operaciones tienen clases específicas para ellas dentro del paquete javax.microedition.rms y a continuación se van a ver más detalladamente.

Operaciones avanzadas con Record Stores

A través de un Record store se puede navegar por medio de un bucle, pero esto no es nada efectivo, requiere demasiado consumo de recursos para realizar las operaciones. Por ello el paquete RMS proporciona la interfaz RecordEnumeration que facilita esta tarea de navegación. En la siguiente tabla aparecen los métodos que componen esta interfaz.

Método	Descripción
int numRecords()	Devuelve el número de registros
Byte[] nextRecord()	Devuelve el siguiente registro
int nextRecordId()	Devuelve el siguiente 'id' a devolver
Byte[] previousRecord()	Devuelve el registro anterior
int previousRecordId()	Devuelve el 'id' del registro anterior
boolean hasNextElement()	Devuelve true si existen más registros en adelante
boolean hasPreviousElement()	Devuelve true si existen más registros anteriores
void keepUpdated()	Provoca que los índices se actualicen cuándo se produzca algún cambio en el Record Store
boolean isKeptUpdated()	Devuelve true si los índices se actualizan al producirse algún cambio en el RecordStore
void rebuild()	Actualiza los índices del RecordEnumeration
void reset()	Actualiza los índices a su estado inicial
void destroy()	Libera todos los recursos ocupados

Como se puede ver en la anterior tabla, la navegación por los registros usando RecordEnumeration es mucho mas intuitiva, ya que es posible moverte hacia delante o hacia atrás con la llamada a un método. Se pueden actualizar índices en cualquier momento, ver el número de registros que componen el Record Store, etc…

Usando esta interfaz, como se ha dicho anteriormente, se podría sustituir un bucle optimizando recursos y aprovechando la potencia de esta herramienta. La sustitución seria algo como lo siguiente:

```
RecordEnumeration re = rs.enumerateRecords(null,null,false);
while (re.hasNextElement()){
registro = re.nextRecord();
//Aquí irían las operaciones a realizar
…
}
```

Como se puede observar se ha creado un objeto RecordEnumeration utilizando el método enumerateRecords, a este se le ha pasado como primer parámetro (RecordFilter) null, como segundo parámetro (RecordComparator) también null y el tercer parámetro es false porque no interesa que se actualicen los índices en el caso de que se produzca alguna acción en el Record Store. Las interfaces RecordFilter y RecordComparator serán vistas a continuación pero podemos adelantar que el primero es utilizado para realizar un filtrado de records y el segundo para comparar diversos records entre ellos.

Búsqueda de registros: para realizar una búsqueda de registros eficiente dentro de un Record Store se puede utilizar la interfaz RecordFilter. Esta se encarga de devolver al RecordEnumeration únicamente los registros que coincidan con un determinado patrón de búsqueda. Para poder utilizar esta interfaz se deberá implementar obligatoriamente el método: *public boolean matches (byte [] candidato)*, ya que este es el único método abstracto de dicha interfaz. Este método se encarga de comparar el registro candidato pasado como parámetro con el valor que queremos buscar y devolver true si encuentra alguna coincidencia. Para verlo con una mayor claridad a continuación se codificará un RecordFilter con su método matches:

```
public class Filtro implements RecordFilter{
 private String cadenaabuscar = null;

 public Filtro(String cadena){
       this.cadenaabuscar = cadena.toLowerCase();
 }

 public boolean matches(byte[] candidato){
       boolean resultado = false;
       String cadenacandidata;
       ByteArrayInputStream bais;
       DataInputStream dis;
       try{
             bais = new ByteArrayInputStream(candidato);
             dis = new DataInputStream(bais);
             cadenacandidata = dis.readUTF().toLowerCase();
             }
       catch (Exception e){
       return false;
       }
       if ((cadenacandidata != null) &&
 (cadenacandidata.indexOf(cadenaabuscar)!= -1))
             return true;
       else

             return false;

 }
}
```

Si usamos este RecordFilter a la hora de invocar a enumerateRecords():

```
//Creo un objeto de tipo RecordFilter
Filtro buscar = new Filtro("Antonio");

//Obtengo el RecordEnumeration usando el filtro anterior
RecordEnumeration re = rs.enumerateRecords(buscar,null,false);

//Si encuentro algún registro dado el patrón de búsqueda
if (re.numRecords() > 0)
System.out.println("Patron 'Antonio' encontrado");
```

El resultado será que el RecordEnumeration devuelto sólo contendrá los registros en los que se haya encontrado el patrón de búsqueda.

Ordenación de Registros: para ordenar los registros dentro de un Record Store existe la interfaz RecordComparator, esta se encarga de comparar un registro pasado como parámetro con los pertenecientes al Record Store y según como se quiera ordenar lo ira ubicando dentro del mismo. Esta interfaz también posee un método abstracto de implementación obligatoria llamado:

public int compare (byte[] reg1, byte[] reg2)

Este método se encarga de realizar la comparación entre los campos que se deseen de los registros. Devuelve un entero que indica si reg1 va antes o después que reg2. El valor devuelto por este método puede ser:

- EQUIVALENT: Los registros pasados al método compare son equivalentes.
- FOLLOWS: El primer registro sigue al segundo.
- PRECEDES: El primer registro precede al segundo.

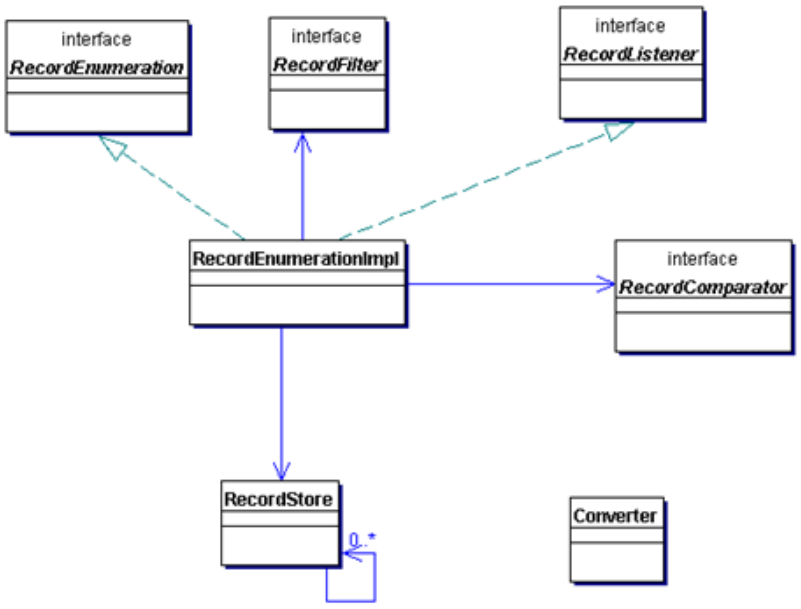


Figura 79. Diagrama de la interfaz RecordEnumeration.

Al igual que en el caso anterior, se creará una clase para comprobar el funcionamiento de esta interfaz y así poder entenderla mas claramente sobre un caso práctico.

```
public class Compara implements RecordComparator{

 public boolean compare(byte[] reg1, byte[] reg2){

       ByteArrayInputStream bais;
       DataInputStream dis;
       String cad1, cad2;

       try{
             bais = new ByteArrayInputStream(reg1);
             dis = new DataInputStream(bais);
             cad1 = dis.readUTF();

             bais = new ByteArrayInputStream(reg2);
             dis = new DataInputStream(bais);
             cad2 = dis.readUTF();

             int resultado = cad1.compareTo(cad2);

             if (resultado == 0)
                   return RecordComparator.EQUIVALENT;
             else if (result < 0)
                   return RecordComparator.PRECEDES;
             else
                   return RecordComparator.FOLLOWS;
       }

       catch (Exception e){

             return RecordComparator.EQUIVALENT;
       }
 }
}
```

Esta clase puede usarse a la hora de crear el RecordEnumeration para recorrer los elementos de manera ordenada:

```
//Creo un objeto de tipo RecordComparator
    Compara comp = new Compara();

//Obtengo el RecordEnumeration usando el objeto anterior, y con los registros ordenados
    RecordEnumeration re = rs.enumerateRecords(null,comp,false);
```

Manejo de eventos en los Record Stores: La tercera de las interfaces especiales a las que se ha hecho mención anteriormente es esta. Se llama RecordListener y es la encargada de recoger los eventos generados por los records. Esta interfaz funciona como cualquier otro listener en java, cuando ocurre algún evento, un método es llamado para notificar el cambio. Se compone de tres métodos para manejar todos los eventos, estos métodos son los siguientes:

- *public void recordAdded(RecordStore rs, int id),* se invoca cuando un registro es añadido al Record Store.
- *public void recordChanged(RecordStore rs, int id),* se invoca cuando un registro es modificado.
- *public void recordDeleted(RecordStore rs, int id),* se invoca cuando un registro es borrado.

La implementación de estos métodos es vacía, por lo tanto para su utilización deberá ser implementada por el programador. Ahora como en las anteriores interfaces se creara un ejemplo de uso de estos métodos.

```
public class PruebaListener implements RecordListener{

 public void recordAdded(RecordStore rs, int id){
       try{ String nombre = rs.getName();
             System.out.println("Registro "+id+" añadido al Record
       Store: "+nombre);
       }
       catch (Exception e){
             System.err.println(e);
       }
 }

 public void recordDeleted(RecordStore rs, int id){
       try{ String nombre = rs.getName();
             System.out.println("Registro "+id+" borrado del Record
       Store: "+ nombre);
       }
       catch (Exception e){
             System.err.println(e);
       }
 }

 public void recordChanged(RecordStore rs, int id){
       try{String nombre = rs.getName();
             System.out.println("Registro "+id+" modificado del
       Record Store: "+nombre);
       }
       catch (Exception e){
             System.err.println(e);
       }
 }
}
```

Añadir este RecordListener a un Record Store es muy sencillo. Sólo tenemos que incluir el siguiente código en el MIDlet:

```
RecordStore rs = openRecordStore ("Almacen1",true);
rs.addRecordListener( new PruebaListener());
```

Con esta interfaz se termina la explicación de este capitulo, ya que ya se han visto todos los métodos e interfaces que posee la clase Record Store. Con esto se ha aprendido a almacenar cualquier tipo de información de manera no volátil en el dispositivo MID. Se procederá por tanto a el desarrollo de un ejemplo práctico donde poder poner en uso todas las técnicas aprendidas en este capitulo.

3.3.2.3 Codificación de una Aplicación en modo Agenda

Para el desarrollo de esta aplicación se hará uso de la herramienta anteriormente explicada Netbeans, y del paquete RMS. El objetivo es crear una agenda telefónica en la cual poder guardar en la memoria no volátil del MID el nombre, apellidos y número de teléfono de personas. A continuación se pondrá todo el código creado para tal efecto y después se harán las explicaciones pertinentes acompañadas por imágenes de su emulación en el ordenador.

Esta aplicación se compone de dos paquetes, el primero de ellos contendrá los códigos fuente de la aplicación, el segundo contiene unas imágenes .png para atribuir una mejor imagen final a la interfaz.

El primero de estos paquetes se compone de tres clases:

- Memo.java: esta es el MIDlet en si, es la clase que mueve a las demás, se encarga de realizar las llamadas a las otras clases para hacer que la aplicación funcione correctamente (hereda de MIDlet).

- Rms.java: En esta clase se hace el manejo de todos los records y del Record Store. Se encarga de agregar los records al listín, devolver los datos a la clase Memo, etc…

- ShowCanvas.java: esta tercera clase es la encargada de la interfaz. Ubica todos los elementos en sus respectivos lugares y otorga la apariencia final a la aplicación (hereda de Canvas, ya se debe conocer esta clase de Java).

El código de Memo.java es el siguiente:

```
package Memo;

  import javax.microedition.midlet.*;
  import javax.microedition.lcdui.*;
  import javax.microedition.rms.*;
  import javax.microedition.io.*;

   public class Memo extends MIDlet implements CommandListener {

      private Command exitCommand;
      private Command xitCommand;
      private Command tCommand;
      private Command fontCommand;
      private Command backCommand;
      private Command saveCommand;
      private Command displayCommand;

      private Display display;
      private TextBox t = null;
```

```
        private TextBox t2 = null;
        private List Selectfont;
        private String[] options1={"Fuente Tipo 1", "Fuente Tipo 2",
        "Fuente Tipo 3", "Fuente Tipo 4", "Fuente Tipo 5", "Fuente Tipo
        6"};
        private Form mainscreen;
        private Form subscreen1;
        private Form subscreen2;

        private Rms myRms = new Rms();

        static final String REC_STORE = "text_db";

        // Declaro el objeto creado
        public Memo() {

              display = Display.getDisplay(this);
              exitCommand = new Command("Salir", Command.EXIT, 2);
              fontCommand = new Command("Elegir fuente", Command.SCREEN,
              1);
              xitCommand = new Command("Añadir nuevo", Command.SCREEN, 2);
              tCommand = new Command("Comprobar listin", Command.SCREEN,
              2);
              backCommand = new Command ("Atras", Command.BACK, 1);
              saveCommand = new Command ("Guardar", Command.SCREEN, 1);
              displayCommand = new Command ("Mostrar fecha",
              Command.SCREEN, 2);


              Selectfont = new List(" Selecciona fuente !", List.IMPLICIT,
options1, null);
              Selectfont.setCommandListener(this);
        }

        // Inicio el MIDlet creando el Textbox y
        // asociando el comando salir y los escuchadores

        public void startApp() {

              mainscreen = new Form ( " Memoria");

              Image img = null;

              try {
                    img = Image.createImage("/icons/mountains.png");
              }

              catch (Exception e) {}

              mainscreen.append(new StringItem ("", "Bienvenido"));
              mainscreen.append(new ImageItem(" ",img,
              ImageItem.LAYOUT_CENTER, "La imagen no puede ser mostrada"));

              mainscreen.addCommand(exitCommand);
              mainscreen.addCommand(fontCommand);
              mainscreen.addCommand(xitCommand);
              mainscreen.addCommand(tCommand);
              mainscreen.addCommand(displayCommand);

              mainscreen.setCommandListener(this);
              myRms.openRecStore(REC_STORE);
```

```
        display.setCurrent(mainscreen);
}

public void pauseApp() { }

public void destroyApp(boolean unconditional) { }

public void commandAction(Command c, Displayable s) {


        if (c == exitCommand) {
                destroyApp(false);
                notifyDestroyed();
        }

        if (c == fontCommand) {
              Selectfont.addCommand(backCommand);
              Selectfont.setCommandListener(this);
              display.setCurrent(Selectfont);
              }

        if(c==backCommand)
              display.setCurrent(mainscreen);

               if ( s == Selectfont && c == List.SELECT_COMMAND) {
                    t2 = new TextBox("Seleccione fuente",
                    options1[((List)s).getSelectedIndex()] + " es
                    seleccionada" , 256, 0);
                    display.setCurrent(t2);
                    t2.addCommand(backCommand);
                    t2.setCommandListener(this);

                 }

              if (c == xitCommand){

        t = new TextBox("Please input text", "", 256, 0);
        t.addCommand(saveCommand);
        t.addCommand(backCommand);
        t.setCommandListener(this);
        display.setCurrent(t);

              }

              if (c == tCommand){

              ShowCanvas myCanvas = new
              ShowCanvas(Selectfont.getSelectedIndex(), myRms);
                    myCanvas.addCommand(backCommand);
                    myCanvas.setCommandListener(this);

                    display.setCurrent(myCanvas);

              }

        if (c==displayCommand){

              Image img2 = null;
              try {
                    img2 = Image.createImage("/icons/sun.png");
              }
```

```
                catch (Exception e) {}


                subscreen2 = new Form ("Fecha");
                DateField d = new DateField ("La fecha de hoy es :",
                DateField.DATE);
                java.util.Date now = new java.util.Date();
                d.setDate(now);
                subscreen2.append(d);

                subscreen2.append(new ImageItem(" ", img2,
                ImageItem.LAYOUT_CENTER, "La imagen no puede ser
                mostrada"));

                subscreen2.addCommand(backCommand);

                subscreen2.setCommandListener(this);
                display.setCurrent(subscreen2);

        }

        if ( c==saveCommand){
                myRms.writeRecord(t.getString());
                display.setCurrent(mainscreen);
        }
        }
}
```

Esta clase no tiene muchos secretos, se utilizan las herramientas vistas en el capitulo anterior de MIDlets acompañadas por las clases Java de siempre. También se pueden observar las llamadas a las otras clases que componen la aplicación y que se pondrán a continuación.

El método commandAction es el encargado de manejar los eventos generados por la aplicación al pulsar las diferentes teclas del dispositivo. Según el botón pulsado llamará a una clase, a un objeto o a lo que haga falta para responder al evento que se le esta pidiendo ejecutar. Esto no es algo especifico de J2ME ya que anteriormente al programar cualquier aplicación Java siempre ha habido que controlar eventos generados tanto por el ratón como por el teclado, la única diferencia es que en este caso los eventos que se manejan serán proporcionados por el teclado de un dispositivo MID y la manera en que hay que tratarlos es un poco diferente.

También se puede observar que están implementados los tres métodos básicos mencionados en la primera práctica (startApp(), pauseApp(), destroyApp()), esta será una constante en todas las aplicaciones que se generen del tipo MIDlet ya que es una de sus principales características.

La utilización de la clase RMS se comentará con el siguiente código que será puesto a continuación, ya que es la encargada del manejo de los records y del Record Store.

El código de Rms.java es el siguiente:

```
package Memo;

import javax.microedition.rms.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;

public  class Rms {
 public RecordStore recStore = null;
 public int number_reg = 0;
 public String str[]= null;

public void Rms () {
}

public void openRecStore(String Rec_Store){
      try{
            recStore = RecordStore.openRecordStore(Rec_Store, true);
       }
      catch (Exception e){System.err.println(e.toString());
      }
}

public void writeRecord(String str)
  {
      byte[] rec = str.getBytes();
      try{
            recStore.addRecord(rec, 0, rec.length);
      }
      catch (Exception e){
            System.err.println(e.toString());
       }
  }

public void readRecords() {
   int number = 0;
   try {
      System.out.println("Número de records:" +
recStore.getNumRecords());
         int id;
         str = new String[recStore.getNumRecords()];
         if (recStore.getNumRecords() > 0)
         {
            System.out.println(str.length);
            RecordEnumeration re = recStore.enumerateRecords(null, null,
false);

            while (re.hasNextElement())
            {
                  str[number] = new String (re.nextRecord());
                  number++;
      }
            }
       }catch (Exception e){System.err.println(e.toString());}

      number_reg = str.length;

   }
}
```

Esta clase se compone de 3 métodos básicos además del constructor de la clase. La función de cada uno de ellos es la siguiente:

- *public void openRecStore(String RecStore):*este método se encarga de abrir el Record Store para que se puedan insertar en el mismo nuevos records. Funciona de una manera muy sencilla, solo hay que introducirle como parámetro el nombre del Record Store con el que se quiera trabajar, si este nombre ya existe simplemente lo abre. Si no existe ningún Record Store con el nombre que se le ha pasado, crea un nuevo Record Store con el nombre elegido.

- *public void writeRecord (String str):* método utilizado para añadir nuevos records al Record Store. Se le pasa como parámetro otro String que será la información que se quiere guardar en el record (esta información puede contener caracteres y números), el método extrae la información pasándola a formato bytes y una vez hecho esto añade el record. Hay que saber que en RMS es obligatorio el uso de try y catch para poder encontrar posibles excepciones.

- *public void readRecords():* esta clase se encarga de leer todos los records contenidos en el Record Store para poder mostrarlos en el listín. Se compone de un bucle while que va extrayendo todos los records y pasándoselos al método principal.

La ultima clase que queda por mostrar y por comentar es la clase ShowCanvas.java encargada de la apariencia de la aplicación, esta clase no tiene mucho que comentar ya que hereda de Canvas y utiliza sus métodos. Estos métodos ya son muy conocidos por todo el mundo puesto que Java los integra desde que se creo y todo el mundo ha debido trabajar con ellos alguna vez. Por tanto se pondrá la clase a continuación y habrá poco más que comentar de la misma.

Código de la clase ShowCanvas.java.

```
package Memo;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.microedition.io.*;


public class ShowCanvas extends Canvas {

Font font = null;
Rms myRms = null;

  public ShowCanvas(int _index, Rms _myRms) {
      setFont(_index);
      setRms(_myRms);

  }

  public void setRms(Rms _myRms) {
      myRms = _myRms;
  }
```

```
  public void setFont(int index ) {
      switch( index ){
          case 0:
                   font = Font.getFont (Font.FACE_SYSTEM,
            Font.STYLE_PLAIN, Font.SIZE_SMALL);
            break;
          case 1:
             font = Font.getFont (Font.FACE_MONOSPACE,  Font.STYLE_BOLD,
            Font.SIZE_MEDIUM);
            break;
          case 2:
            font = Font.getFont (Font.FACE_PROPORTIONAL,
            Font.STYLE_ITALIC, Font.SIZE_LARGE);
            break;
          case 3:
            font =Font.getFont (Font.FACE_SYSTEM, Font.STYLE_BOLD,
            Font.SIZE_LARGE);
            break;
          case 4:
            font =  Font.getFont (Font.FACE_MONOSPACE, Font.STYLE_ITALIC,
            Font.SIZE_MEDIUM);
            break;
          case 5:
            font = Font.getFont (Font.FACE_PROPORTIONAL,
            Font.STYLE_PLAIN, Font.SIZE_SMALL);
            break;
                default:
            font = Font.getFont (Font.FACE_PROPORTIONAL,
            Font.STYLE_PLAIN, Font.SIZE_SMALL);
                }

  }

  public void paint (Graphics g) {
    int x = 0;
    int y = 0;

    g.setGrayScale (255);
    g.fillRect (0, 0, getWidth (), getHeight ());
    g.setGrayScale (0);

    myRms.readRecords();

         g.setFont(font);
    try {
         for(int t=0;t<myRms.str.length;t++){

         if (myRms.str[t] != null) {
            g.drawString ( myRms.str[t], x, y+1, Graphics.TOP |
Graphics.LEFT);
            System.out.println(myRms.str[t]);
                  y+=font.getHeight () + 1;
                }
           }
     }
    catch (Exception e)
    {
      System.err.println(e.toString());
    }
  }
}
```

Como se puede ver en esta clase que hereda de Canvas tiene implementado el método paint(). Este método obligatorio implementarlo en las clases que heredan de Canvas ya que es el método que define el área de dibujo y lo que se quiere dibujar en ella. También contiene otro método llamado setFont() que es utilizado para elegir el tipo de fuente con el que se quiere mostrar el listín de nombres de la aplicación. Este método contiene la sentencia de elección switch que será combinada en la aplicación con las diferentes posibles elecciones. Por último comentar que posee como la mayoría de las clases un constructor sobrecargado con las variables que hacen falta para la ejecución correcta de esta clase y del resto de la aplicación.

Tal y como se menciono anteriormente, para mejorar la apariencia de la aplicación se agrego un segundo paquete compuesto por unas imágenes. Estas imágenes es importante resaltar que están en formato .png, ya que hay algunos dispositivos que tienen problemas con otros tipos de formatos, este en cambio es aceptado por todos por igual y nunca causa ningún tipo de excepción. Por lo tanto es recomendable utilizarlo en las aplicaciones. Por supuesto pueden utilizarse otros tales como .jpg, .bmp o .png pero quizá haya algún dispositivo MID que no sea capaz de mostrarlo.

Para terminar este capitulo se agregará ahora un conjunto de imágenes correspondientes a la emulación de la aplicación creada.

Figura 80. Captura de pantalla de bienvenida

Figura 81. Menú de la aplicación

Figura 82. Menú de selección del tipo de fuente.

Figura 83. Pantalla para la inserción de datos.

Figura 84. Pantalla mostrando el listín con los datos Guardados.

Figura 85. Pantalla con el modo Mostrar Fecha.

3.3.3 Comunicaciones e Intercambio con Symbian OS

3.3.3.1 Introducción

Uno de los aspectos más fuertes de la telefonía y los dispositivos móviles (en este caso dentro del marco Symbian OS) es la gran cantidad de conexiones que soportan. Un MID hoy por hoy puede conectarse con diversidad de redes utilizando diversas tecnologías, cada una con un uso especifico. Entre ellas se pueden destacar WIFI, Bluetooth, IRda (infrarrojos), Gprs, y un largo etc. Por ello en este capitulo se va a desarrollar una aplicación haciendo uso del Bluetooth de un dispositivo móvil para iniciar un poco el salto a las librerías y paquetes de conexión que se pueden utilizar en J2ME.

Para ello primero se hará una breve exposición de las tecnologías anteriormente mencionadas (WIFI, IRda y Bluetooth) explicando su modo de funcionamiento, características principales y recursos necesarios. Después se hará hincapié en la tecnología Bluetooth que es la que en J2ME proporciona un conjunto de librerías adecuado para su uso. A pesar de esto hay que resaltar la idea de que en J2ME no se puede realizar un aprovechamiento óptimo de esta tecnología la cual se ve beneficiada del uso de otros lenguajes como podría ser C++. En capítulos siguientes se vera como utilizar esta tecnología sobre Symbian OS utilizando como lenguaje C++.

3.3.3.2 WIFI

Wi-Fi es un conjunto de estándares para redes inalámbricas basados en las especificaciones IEEE 802.11. Creado para ser utilizado en redes locales inalámbricas, es frecuente que en la actualidad también se utilice para acceder a Internet.

Wi-Fi es una marca de la *Wi-Fi Alliance* (anteriormente la *WECA: Wireless Ethernet Compatibility Alliance*), la organización comercial que adopta, prueba y certifica que los equipos cumplen los estándares 802.11.

Figura 86. Logotipo de la tecnología Wi-Fi

- ***Historia***

  El problema principal que pretende resolver la normalización es la compatibilidad. No obstante existen distintos estándares que definen distintos tipos de redes inalámbricas. Esta variedad produce confusión en el mercado y descoordinación en los fabricantes. Para resolver este problema, los principales vendedores de soluciones inalámbricas (3com, Airones, Intersil, Lucent Technologies, Nokia y Symbol

Technologies) crearon en 1999 una asociación conocida como WECA (Wireless Ethernet Compatibility Aliance, Alianza de Compatibilidad Ethernet Inalámbrica).

El objetivo de esta asociación fue crear una marca que permitiese fomentar más fácilmente la tecnología inalámbrica y asegurase la compatibilidad de equipos.

De esta forma en abril de 2000 WECA certifica la interoperatibilidad de equipos según la norma IEEE 802.11b bajo la marca Wi-Fi (Wireless Fidelity, Fidelidad Inalámbrica). Esto quiere decir que el usuario tiene la garantía de que todos los equipos que tenga el sello Wi-Fi pueden trabajar juntos sin problemas independientemente del fabricante de cada uno de ellos. Se puede obtener un listado completo de equipos que tienen la certificación Wi-Fi en Alliance - Certified Products.

En el año 2002 eran casi 150 miembros de la asociación WECA. Como la norma 802.11b ofrece una velocidad máxima de transferencia de 11 Mbps ya existen estándares que permiten velocidades superiores, WECA no se ha querido quedar atrás. Por ese motivo, WECA anunció que empezaría a certificar también los equipos IEEE 802.11a de la banda de 5 Ghz mediante la marca Wi-Fi5.

La norma IEEE.802.11 fue diseñada para sustituir a las capas físicas y MAC de la norma 802.3 (Ethernet). Esto quiere decir que en lo único que se diferencia una red Wi-Fi de una red Ethernet, es en la forma como los ordenadores y terminales en general acceden a la red; el resto es idéntico. Por tanto una red local inalámbrica 802.11 es completamente compatible con todos los servicios de las redes locales de cable 802.3 (Ethernet).

- ***Normalización***

Hay tres tipos de Wi-Fi, basado cada uno de ellos en un estándar IEEE 802.11 aprobado. Un cuarto estándar, el 802.11n, está siendo elaborado y se espera su aprobación final para la segunda mitad del año 2007.

  - Los estándares IEEE 802.11b e IEEE 802.11g disfrutan de una aceptación internacional debido a que la banda de 2.4 GHz está disponible casi universalmente, con una velocidad de hasta 11 Mbps y 54 Mbps, respectivamente. Existe también un primer borrador del estándar IEEE 802.11n que trabaja a 2.4 GHz a una velocidad de 108 Mbps. Aunque estas velocidades de 108 Mbps son capaces de alcanzarse ya con el estandar 802.11g gracias a técnicas de aceleramiento que consiguen duplicar la transferencia teórica. Actualmente existen ciertos dispositivos que permiten utilizar esta tecnología, denominados *Pre-N*, sin embargo, no se sabe si serán compatibles ya que el estándar no está completamente revisado y aprobado.

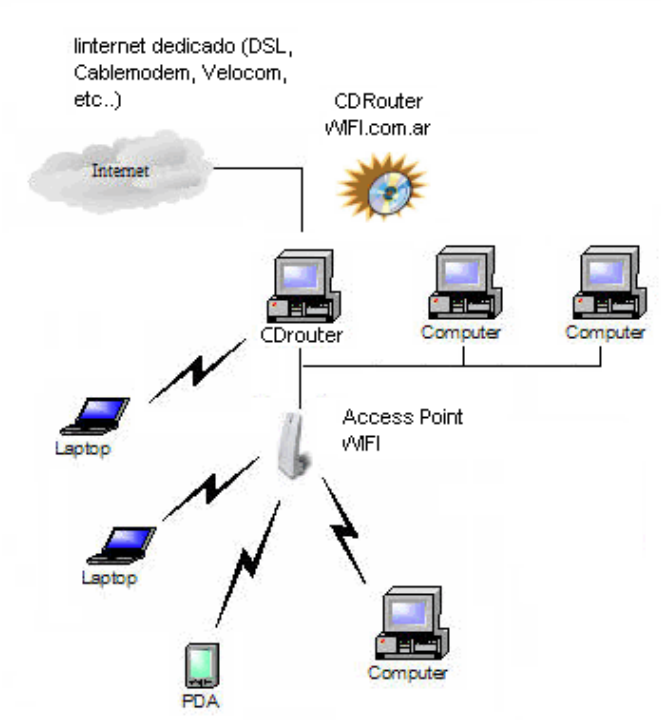


Figura 87. Representación de una red tipo Wi-Fi

- En la actualidad ya se maneja también el estándar IEEE 802.11a, conocido como WIFI 5, que opera en la banda de 5 GHz y que disfruta de una operatividad con canales relativamente limpios. La banda de 5 GHz ha sido recientemente habilitada y, además no existen otras tecnologías (Bluetooth, microondas, etc.) que la estén utilizando, por lo tanto hay muy pocas interferencias...

La tecnología inalámbrica Bluetooth también funciona a una frecuencia de 2.4 GHz por lo que puede presentar interferencias con Wi-Fi, sin embargo, en la versión 1.2 y mayores del estándar Bluetooth se ha actualizado su especificación para que no haya interferencias en la utilización simultánea de ambas tecnologías.

- ***Ventajas y desventajas***

Una de las desventajas que tiene el sistema Wi-Fi es la pérdida de velocidad en relación a la misma conexión utilizando cables, debido a las interferencias y pérdidas de señal que el ambiente puede acarrear. Existen algunos programas capaces de capturar paquetes, trabajando con su tarjeta Wi-Fi en modo promiscuo, de forma que puedan calcular la contraseña de la red y de esta forma acceder a ella, las claves de tipo WEP son relativamente *fáciles de conseguir* para cualquier persona con un conocimiento medio de informática. La alianza Wi-Fi arregló estos problemas sacando el estándar WPA y posteriormente WPA2, basados en el grupo de trabajo 802.11i. Las redes protegidas con WPA2 se consideran robustas dado que proporcionan muy buena seguridad.

Los dispositivos Wi-Fi ofrecen gran comodidad en relación a la movilidad que ofrece esta tecnología, sobre los contras que tiene Wi-Fi es la capacidad de terceras personas para conectarse a redes ajenas si la red no está bien configurada y la falta de seguridad que esto trae consigo.

Cabe aclarar que esta tecnología no es compatible con otros tipos de conexiones sin cables como Bluetooth, GPRS, UMTS, etc.

3.3.3.3 Infrared Data Association

Infrared Data Association (IrDA) define un estándar físico en la forma de transmisión y recepción de datos por rayos infrarrojo. IrDA se crea en 1993 entre HP, IBM, Sharp y otros.

Figura 88. Logotipo IrDA

Esta tecnología, basada en rayos luminosos que se mueven en el espectro infrarrojo. Los estándares IrDA soportan una amplia gama de dispositivos eléctricos, informáticos y de comunicaciones, permite la comunicación bidireccional entre dos extremos a velocidades que oscilan entre los 9.600 bps y los 4 Mbps. Esta tecnología se encuentra en muchos ordenadores portátiles, y en un creciente número de teléfonos celulares, sobre todo en los de fabricantes líderes como Nokia y Ericsson.

El FIR (Fast Infrared) se encuentra en estudio, con unas velocidades teóricas de hasta 16 Mbps.

- ***Estructura***

En IrDA se define una organización en capas:

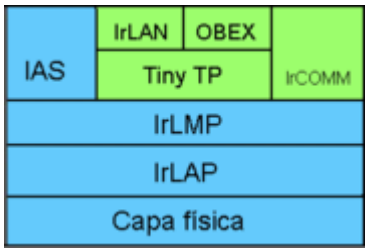


Figura 89. Estructura IrDA

Además cualquier dispositivo que quiera obtener la conformidad de IRDA ha de cumplir los protocolos obligatorios (azul), no obstante puede omitir alguno o todos los protocolos opcionales (verde). Esta diferenciación permite a los desarrolladores optar por diseños más ligeros y menos costosos, pudiendo también adecuarse a requerimientos más exigentes sin que sea necesario salirse del estándar IRDA.

- ***Características***

  - Adaptación compatible con futuros estándares.
  - Cono de ángulo estrecho de 30º .
  - Opera en una distancia de 0 a 1 metro.
  - Conexión universal sin cables.
  - Comunicación punto a punto.
  - Soporta un amplio conjunto de plataformas de hardware y software.

- ***Protocolos IrDA***

  - PHY (Physical Signaling Layer) establece la distancia máxima, la velocidad de transmisión y el modo en el que la información se transmite.

  - IrLAP (Link Access Protocol) facilita la conexión y la comunicación entre dispositivos.

  - IrLMP (Link Management Protocol) permite la multiplexación de la capa IrLAP.
  - IAS (Information Access Service) actúa como unas páginas amarillas para un dispositivo.
  - Tiny TP mejora la conexión y la transmisión de datos respecto a IrLAP.
  - IrOBEX diseñado para permitir a sistemas de todo tamaño y tipo intercambiar comandos de una manera estandarizada.
  - IrCOMM para adaptar IrDA al método de funcionamiento de los puertos serie y paralelo.
  - IrLan permite establecer conexiones entre ordenadores portátiles y LANs de oficina.

Figura 90. Representación de una red tipo IrDA.

- **IrPHY**

La capa física IrPHY establece la distancia máxima, la velocidad de transmisión y el modo en el que se transmite la información. Su segmentación provee servicios de transmisión y recepción para paquetes individuales. La capa más baja de la plataforma IrDA presenta las siguientes especificaciones:

Rango (Estándar: 1 metro; baja-energía a baja-energía: 0,2 metros; Estándar a baja-energía: 0,3 metros). Ángulo (cónico mínimo +- 15°). Velocidad (2.4 kbit/s a 16 Mbit/s). Modulado (Banda Base, sin portadora).

Los transceptores (transmisor-receptor) de IrDA se comunican con pulsos infrarrojos en un cono con medio ángulo de mínimo 15 grados. Las especificaciones de IrPHY requieren un mínimo de irradiación para que la señal pueda ser visible a un metro de distancia, de igual manera requiere que no se exceda un máximo de irradiación para no abrumar un receptor con brillo cuando viene un dispositivo cerca. En el mercado hay dispositivos que no alcanzan un metro, mientras otros pueden alcanzar varios metros, también existen dispositivos que no toleran proximidad extrema. La distancia típica para las comunicaciones IrDA es de 5 a 60 centímetros de separación entre los transceptores, en el medio del cono.

La comunicación IrDA funciona en modo half duplex debido a que su receptor es cegado por la luz de su transmisor, así la comunicación full duplex no es factible. Dos dispositivos simulan conexión full duplex invirtiendo la comunicación rápidamente.

IrPHY se compone de tres especificaciones físicas: SIR (Serial Infrared, Infrarrojo Serial), MIR (Medium Infrared, Infrarrojo Medio) y FIR (Fast Infrared, Infrarrojo Rápido). SIR cubre las velocidades de transmisión soportadas por el puerto RS-232 (9600 bps, 19.2 kbps, 38.4 kbps, 57.6 kbps y 115.2 kbps); dado que el denominador común más bajo para todos los dispositivos es 9600 bps el descubrimiento y la negociación se realizan a esta velocidad. MIR es un término no oficial utilizado para referirse a las velocidades de transmisión de 57.6 kbps a 115.2 kbps. FIR es visto como un término obsoleto por la especificación IrDA, pero no obstante es comúnmente usado para denotar las velocidades de transmisión de 4 Mbps.

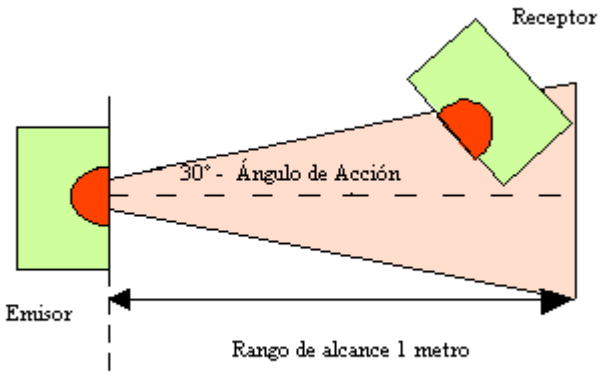


Figura 91. Representación del sistema IrDA

- **IrLAP**

La capa IrLAP (Infrared Link Access Protocol, Protocolo de Acceso al Enlace Infrarrojo) se utiliza para el descubrimiento de dispositivos dentro del rango y el establecimiento de conexiones confiables entre ellos. Es la segunda capa de la especificación IrDA, entre IrPHY e IrLMP y representa la capa de Enlace de Datos del modelo de referencia OSI.

Sus especificaciones más importantes son: Control de acceso Establecimiento de una conexión bidireccional confiable Negociación de los roles primario/secundario de los dispositivos En IrLAP la comunicación de los dispositivos se divide en dispositivos primarios y uno o más dispositivos secundarios. El dispositivo primario controla a los secundarios. Al dispositivo secundario se le permite enviar sólo si el primario se lo solicita.

Las conexiones IrLAP están etiquetadas por el par de las direcciones (32-bits) de los dispositivos envueltos en la conexión. En el siguiente establecimiento de conexión, una dirección de conexión de 7-bits temporal es usada en los paquetes como un alias para esa dirección de dispositivos concatenada.

IrLAP define un esquema de descubrimiento de dispositivo con ranuras de tasa fija que permiten establecer el contacto inicial. Los parámetros de comunicación críticos tales como la tasa de conexión de datos, el máximo tamaño del paquete, el mínimo y máximo intervalo de tiempo, se negocian durante el establecimiento de la conexión. Siguiendo con el establecimiento de la conexión IrLAP, dos dispositivos comprometidos en la comunicación estiman la región espacial que ambos iluminan, literalmente la unión de dos conos solapados de 1m cada uno, con medio ángulo de 15 grados mínimo.

IrLAP provee un mecanismo básico de descubrimiento de dispositivos. Funcionalmente, el resultado de invocar el proceso de descubrimiento IrLAP es una lista de registros que codifican: Dirección del dispositivo: Un identificador de 32-bits semi-permanente del dispositivo descubierto. NickName (Apodo): Un pequeño nombre del dispositivo descubierto que puede ser presentado en las interfaces de usuario para ayudarlo en la selección. Hints (Pistas): Una máscara de bits dando pistas (no oficial) de los servicios que pueden estar disponibles en el dispositivo descubierto. Esto puede ser usado para ordenar consultas en el IAS para establecer concienzudamente la presencia o ausencia de un servicio en particular.

3.3.3.4 Bluetooth

Bluetooth es un estándar de facto global que identifica un conjunto de protocolos que facilitan la comunicación inalámbrica entre diferentes tipos de dispositivos electrónicos. Su nombre viene del rey vikingo, Harald Bluetooth (940 A.D.-981A.D.), famoso por su habilidad para la comunicación, y para hacer que la gente hablara entre ella.

Figura 92. Logotipo Bluetooth

Introducción

Bluetooth es una tecnología de radio de corto alcance, que permite conectividad inalámbrica entre dispositivos remotos. Se diseñó pensando básicamente en tres objetivos: pequeño tamaño, mínimo consumo y bajo precio.

Nociones sobre Bluetooth

Bluetooth opera en la banda libre de radio ISM1 a 2.4 Ghz. Su máxima velocidad de transmisión de datos es de 1 Mbps. El rango de alcance Bluetooth depende de la potencia empleada en la transmisión. La mayor parte de los dispositivos que usan Bluetooth transmiten con una potencia nominal de salida de 0 dBm, lo que permite un alcance de unos 10 metros en un ambiente libre de obstáculos.

- **Salto de frecuencia:**

Debido a que la banda ISM está abierta a cualquiera, el sistema de radio Bluetooth deberá estar preparado para evitar las múltiples interferencias que se pudieran producir. Éstas pueden ser evitadas utilizando un sistema que busque una parte no utilizada del espectro o un sistema de salto de frecuencia.

En este caso la técnica de salto de frecuencia es aplicada a una alta velocidad y una corta longitud de los paquetes (1600 saltos/segundo). Con este sistema se divide la banda de frecuencia en varios canales de salto, donde, los transceptores, durante la conexión van cambiando de uno a otro canal de salto de manera pseudo-aleatoria.

Los paquetes de datos están protegidos por un esquema ARQ (repetición automática de consulta), en el cual los paquetes perdidos son automáticamente retransmitidos.

- **El canal:**

Bluetooth utiliza un sistema FH/TDD (salto de frecuencia/división de tiempo duplex), en el que el canal queda dividido en intervalos de 625 µs, llamados *slots*, donde cada salto de frecuencia es ocupado por un *slot*.

Dos o más unidades Bluetooth pueden compartir el mismo canal dentro de una *piconet* (pequeña red que establecen automáticamente los terminales Bluetooth para comunicarse entre si), donde una unidad actúa como maestra, controlando el tráfico de datos en la *piconet* que se genera entre las demás unidades, donde éstas actúan como esclavas, enviando y recibiendo señales hacia el maestro.

El salto de frecuencia del canal está determinado por la secuencia de la señal, es decir, el orden en que llegan los saltos y por la fase de esta secuencia. En Bluetooth, la secuencia queda fijada por la identidad de la unidad maestra de la *piconet* (un código único para cada equipo), y por su frecuencia de reloj.

- **Datagrama Bluetooth:**

La información que se intercambia entre dos unidades Bluetooth se realiza mediante un conjunto de *slots* que forman un paquete de datos. Cada paquete comienza con un código de acceso de 72 bits, que se deriva de la identidad maestra, seguido de un paquete de datos de cabecera de 54 bits. Éste contiene importante información de control, como tres bits de acceso de dirección, tipo de paquete, bits de control de flujo, bits para la retransmisión automática de la pregunta, y chequeo de errores de campos de cabecera. La dirección del dispositivo es en forma hexadecimal. Finalmente, el paquete que contiene la información, que puede seguir al de la cabecera, tiene una longitud de 0 a 2745 bits.

En cualquier caso, cada paquete que se intercambia en el canal está precedido por el código de acceso. Los receptores de la *piconet* comparan las señales que reciben con el código de acceso, si éstas no coinciden, el paquete recibido no es considerado como válido en el canal y el resto de su contenido es ignorado.

Figura 93. Datagrama Bluetooth

- **Piconets:**

Como hemos citado anteriormente si un equipo se encuentra dentro del radio de cobertura de otro, éstos pueden establecer conexión entre ellos. Cada dispositivo tiene una dirección única de 48 bits, basada en el estándar IEEE 802.11 para WLAN. En principio sólo son necesarias un par de unidades con las mismas características de hardware para establecer un enlace. Dos o más unidades Bluetooth que comparten un mismo canal forman una *piconet*.

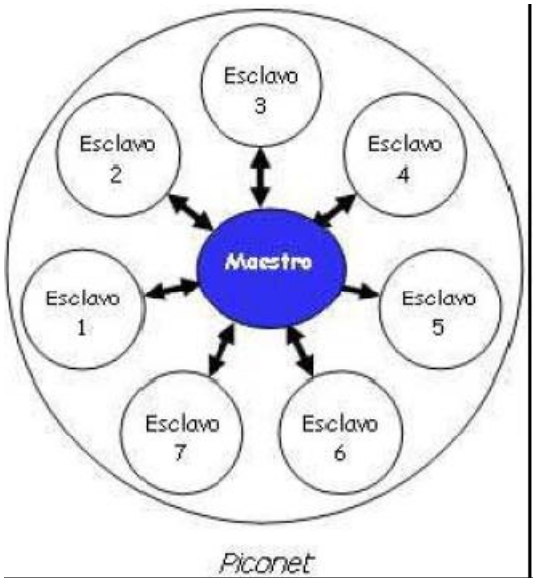


Figura 94. Representación de Piconet

Para regular el tráfico en el canal, una de las unidades participantes se convertirá en maestra, pero por definición, la unidad que establece la *piconet* asume éste papel y todos los demás serán esclavos. Los participantes podrían intercambiar los papeles si una unidad esclava quisiera asumir el papel de maestra. Sin embargo sólo puede haber un maestro en la *piconet* al mismo tiempo. Hasta ocho usuarios o dispositivos pueden formar una *piconet* y hasta diez *piconets* pueden coexistir en una misma área de cobertura.

- **Medios y velocidades:**

Además de los canales de datos, están habilitados tres canales de voz de 64 kbit/s por *piconet*. Las conexiones son uno a uno con un rango máximo de diez metros, aunque utilizando amplificadores se puede llegar hasta los 100 metros, pero en este caso se introduce alguna distorsión. Los datos se pueden intercambiar a velocidades de hasta 1 Mbit/s. El protocolo bandabase que utiliza Bluetooth combina las técnicas de circuitos y paquetes para asegurar que los paquetes llegan en orden.

- **Scatternet:**

Los equipos que comparten un mismo canal sólo pueden utilizar una parte de su capacidad. Aunque los canales tienen un ancho de banda de un 1Mbit, cuantos más usuarios se incorporan a la *piconet*, disminuye la capacidad hasta unos 10 kbit/s más o menos. Para poder solucionar este problema se adoptó una solución de la que nace el concepto de *scatternet*.

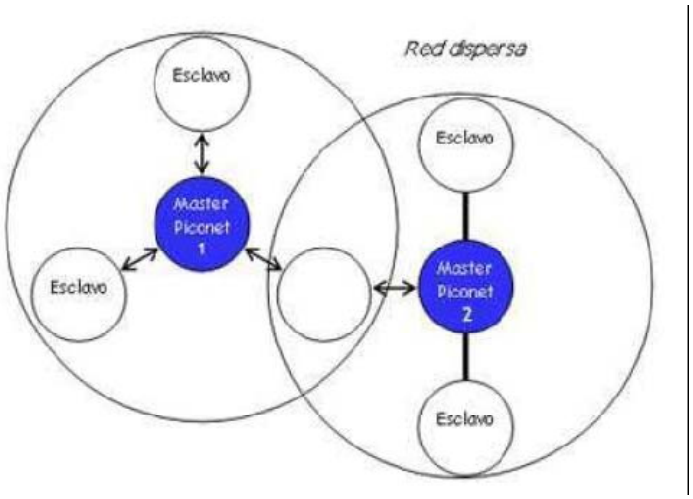


Figura 95. Representación de una Red dispersa

Las unidades que se encuentran en el mismo radio de cobertura pueden establecer potencialmente comunicaciones entre ellas. Sin embargo, sólo aquellas unidades que realmente quieran intercambiar información comparten un mismo canal creando la *piconet*. Este hecho permite que se creen varias *piconets* en áreas de cobertura superpuestas.

A un grupo de *piconets* se le llama *scatternet*. El rendimiento, en conjunto e individualmente de los usuarios de una *scatternet* es mayor que el que tiene cada usuario cuando participa en un mismo canal de 1 Mbit. Además, estadísticamente se obtienen ganancias por multiplexación y rechazo de canales salto. Debido a que individualmente cada *piconet* tiene un salto de frecuencia diferente, diferentes *piconets* pueden usar simultáneamente diferentes canales de salto.

Establecimiento de la conexión

La conexión con un dispositivo, se hace mediante un mensaje *page.* Si la dirección es desconocida, antes del mensaje *page* se necesitara un mensaje *inquiry.* Antes de que se produzca ninguna conexión, se dice que todos los dispositivos están en modo *standby.*

Un dispositivo en modo *standby* se despierta cada 1.28 segundos para escuchar posibles mensajes *page/inquiry*. Cada vez que un dispositivo se despierta, escucha una de las 32 frecuencias de salto definidas. Un mensaje de tipo *page*, será enviado en 32 frecuencias diferentes. Primero el mensaje es enviado en las primeras 16 frecuencias (128 veces), y si no se recibe respuesta, el maestro mandará el mensaje *page* en las 16 frecuencias restantes (128 veces). El tiempo máximo de intento de conexión es de 2.56 segundos.

En el estado conectado, el dispositivo Bluetooth puede encontrarse en varios modos de operación:

- *Active mode*: En este modo, el dispositivo Bluetooth participa activamente en el canal.
- *Sniff mode*: En este modo, el tiempo de actividad durante el cual el dispositivo esclavo escucha se reduce. Esto significa que el maestro sólo puede iniciar una transmisión en unos *slots* de tiempo determinados.
- *Hold mode*: En el estado conectado, el enlace con el esclavo puede ponerse en espera. Durante este modo, el esclavo puede hacer otras cosas, como escanear en busca de otros dispositivos, atender otra *piconet*, etc.
- *Park mode*: En este estado, el esclavo no necesita participar en la *piconet*, pero aún quiere seguir sincronizado con el canal. Deja de ser miembro de la *piconet*. Esto es útil por si hay más de siete dispositivos que necesitan participar ocasionalmente en la *piconet*.

APIs Java para Bluetooth

Mientras que el hardware Bluetooth había avanzado mucho, hasta hace relativamente poco no había manera de desarrollar aplicaciones java Bluetooth – hasta que apareció JSR 82, que estandarizó la forma de desarrollar aplicaciones Bluetooth usando Java. Ésta esconde la complejidad del protocolo Bluetooth detrás de unos APIs que permiten centrarse en el desarrollo en vez de los detalles de bajo nivel del Bluetooth.

Estos APIs para Bluetooth están orientados para dispositivos que cumplan las siguientes características:

• Al menos 512K de memoria libre (ROM y RAM) (las aplicaciones necesitan memoria adicional).
• Conectividad a la red inalámbrica Bluetooth.
• Que tengan una implementación del J2ME CLDC.

- **JSR 82:**

El objetivo de ésta especificación era definir un API estándar abierto, no propietario que pudiera ser usado en todos los dispositivos que implementen J2ME. Por consiguiente fue diseñado usando los APIs J2ME y el entorno de trabajo CLDC/MIDP. Los APIs JSR 82 son muy flexibles, ya que permiten trabajar tanto con aplicaciones nativas Bluetooth como con aplicaciones Java Bluetooth. El API intenta ofrecer las siguientes capacidades:

• Registro de servicios.
• Descubrimiento de dispositivos y servicios.
• Establecer conexiones RFCOMM, L2CAP y OBEX entre dispositivos.
• Usar dichas conexiones para mandar y recibir datos (las comunicaciones de voz no están soportadas).
• Manejar y controlar las conexiones de comunicación.
• Ofrecer seguridad a dichas actividades.

Los APIs Java para Bluetooth definen dos paquetes que dependen del paquete CLDC javax.microedition.io:

• javax.bluetooth
• javax.obex

Ahora se verán los métodos que puede usar cada objeto importante para programar aplicaciones que hagan uso de Bluetooth. Se irán viendo las diferentes clases con los métodos que contienen y después se hará uso de algunos de ellos para crear una aplicación no muy compleja cuya tarea será la de encontrar todos los dispositivos Bluetooth que se encuentren dentro de su rango de acción y decir que servicios son capaces de proporcionar cada uno de ellos.

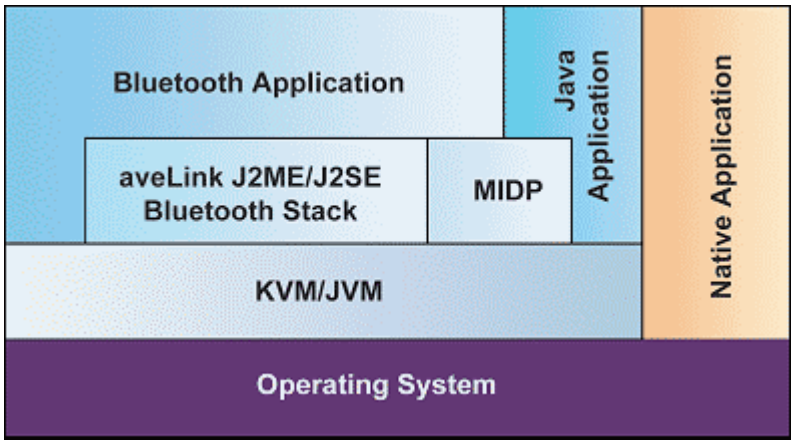


Figura 96. Estructura de la plataforma Bluetooth (JSR 82)

BCC (Bluetooth Control Center)

Los dispositivos Bluetooth que implementen este API pueden permitir que múltiples aplicaciones se estén ejecutando concurrentemente. El BCC previene que una aplicación pueda perjudicar a otra. El BCC es un conjunto de capacidades que permiten al usuario resolver peticiones conflictivas de aplicaciones definiendo unos valores específicos para ciertos parámetros de la pila Bluetooth.

El BCC puede ser una aplicación nativa, una aplicación en un API separado, o sencillamente un grupo de parámetros fijados por el proveedor que no pueden ser cambiados por el usuario. Hay que destacar, que el BCC no es una clase o un interfaz definido en esta especificación, pero es una parte importante de su arquitectura de seguridad.

- **Inicialización de la pila**

  La pila Bluetooth es la responsable de controlar el dispositivo Bluetooth, por lo que es necesario inicializarla antes de hacer cualquier otra cosa. El proceso de inicialización consiste en un número de pasos cuyo propósito es dejar el dispositivo listo para la comunicación inalámbrica.

  Desafortunadamente, la especificación deja la implementación del BCC a los vendedores, y cada vendedor maneja la inicialización de una manera diferente. En un dispositivo puede haber una aplicación con un interfaz GUI, y en otra puede ser una serie de configuraciones que no pueden ser cambiados por el usuario.

Discovery

Dado que los dispositivos inalámbricos son móviles, necesitan un mecanismo que permita encontrar, conectar, y obtener información sobre las características de dichos dispositivos.

Descubrir Dispositivos (Device discovery)

Cualquier aplicación puede obtener una lista de dispositivos a los que es capaz de encontrar, usando, o bien startInquiry() o retrieveDevices() (bloqueante).

startInquiry() requiere que la aplicación tenga especificado un *listener,* el cual es notificado cuando un nuevo dispositivo es encontrado después de haber lanzado un proceso de búsqueda. Por otra parte, si la aplicación no quiere esperar a descubrir dispositivos (o a ser descubierta por otro dispositivo) para comenzar, puede utilizar retrieveDevices(), que devuelve una lista de dispositivos encontrados en una búsqueda previa o bien unos que ya conozca por defecto.

Clases del Device Discovery:

interface javax.bluetooth.DiscoveryListener: Este interfaz permite a una aplicación especificar un evento en el *listener* que reaccione ante eventos de búsqueda. También se usa para encontrar dispositivos.

El método deviceDiscovered() se llama cada vez que se encuentra un dispositivo en un proceso de búsqueda. Cuando el proceso de búsqueda se ha completado o cancelado, se llama al método inquiryCompleted(). Este método recibe un argumento, que puede ser INQUIRY_COMPLETED, INQUIRY_ERROR o INQUIRY_TERMINATED, dependiendo de cada caso.

interface javax.bluetooth.DiscoveryAgent: Esta interfaz provee métodos para descubrir dispositivos y servicios. Para descubrir dispositivos, esta clase provee del método startInquiry() para poner al dispositivo en modo de búsqueda, y el método retrieveDevices() para obtener la información de dispositivos previamente encontrados. Además provee del método cancelInquiry() que permite cancelar una operación de búsqueda.

Descubrir Servicios (Service Discovery)

Ahora se verá la parte del API que usa el cliente para descubrir servicios disponibles en los dispositivos servidores encontrados. La clase DiscoveryAgent provee de métodos para buscar servicios en un dispositivo servidor Bluetooth e iniciar transacciones entre el dispositivo y el servicio. Este API no da soporte para buscar servicios que estén ubicados en el propio dispositivo.

Para descubrir los servicios disponibles en un dispositivo servidor, el cliente primero debe recuperar un objeto que encapsule funcionalidad SDAP (Service Discovery Application Profile) que es igual a (SDP (Service Discovery Profile) + GAP (Generic Access Profile)).

Este objeto es del tipo DiscoveryAgent, cuyo pseudocódigo viene dado por:

DiscoveryAgent da = LocalDevice.getLocalDevice().getDiscoveryAgent();

Clases del Service Discovery:

class javax.bluetooth.UUID: Esta clase encapsula enteros sin signo que pueden ser de 16, 32 ó 128 bits de longitud. Estos enteros se usan como un identificador universal cuyo valor representa un atributo del servicio. Sólo los atributos de un servicio representados con UUID están representados en la Bluetooth SDP.

class javax.bluetooth.DataElement: Esta clase contiene varios tipos de datos que un atributo de servicio Bluetooth puede usar. Algunos de estos son:

- String
- Boolean
- UUID
- Enteros con signo y sin signo, de uno, dos, cuatro o seis bytes de longitud
- Secuencias de cualquiera de los tipos anteriores.

Esta clase además presenta una interfaz que permite construir y recuperar valores de un atributo de servicio.

class javax.bluetooth.DiscoveryAgent: Esta clase provee métodos para descubrir servicios y dispositivos.

interface javax.bluetooth.ServiceRecord: Este interfaz define el Service Record de Bluetooth, que contiene los pares (atributo ID, valor). El atributo ID es un entero sin signo de 16 bits, y valor es de tipo DataElement. Además, este interfaz tiene un método populateRecord() para recuperar los atributos de servicio deseados (pasando como parámetro al método el ID del atributo deseado).

interface javax.bluetooth.DiscoveryListener: Este interfaz permite a una aplicación especificar un *listener* que responda a un evento del tipo Service Discovery o Device Discovery. Cuando un nuevo servicio es descubierto, se llama al método servicesDiscovered(), y cuando la transacción ha sido completada o cancelada se llama a serviceSearchCompleted().

Registro del Servicio (Service Registration)

Las responsabilidades de una aplicación servidora de Bluetooth son:

1. Crear un Service Record que describa el servicio ofrecido por la aplicación.
2. Añadir el Service Record al SDDB del servidor para avisar a los clientes potenciales de este servicio.
3. Registrar las medidas de seguridad Bluetooth asociadas a un servicio.
4. Aceptar conexiones de clientes que requieran el servicio ofrecido por la aplicación.
5. Actualizar el Service Record en el SDDB del servidor si las características del servicio cambian.
6. Quitar o deshabilitar el Service Record en el SDDB del servidor cuando el servicio no está disponible.

A las tareas 1,2, 5 y 6 se las denominan registro del servicio (Service Registration), que comprenden unas tareas relacionadas con advertir al cliente de los servicios disponibles.

Responsabilidades del Registro de Servicio:

Cuando una aplicación llama a Connector.open() con un String conexión URL, la implementación crea un ServiceRecord. El correspondiente registro del servicios añadido a la SDDB por la implementación cuando la aplicación servidora llama a acceptAndOpen(). La aplicación servidora puede acceder a dicho ServiceRecord llamando a getRecord() y hacer las modificaciones pertinentes. Las modificaciones se hacen también en el ServiceRecord de la SDDB cuando la aplicación llama a updateRecord(). Finalmente el ServiceRecord es eliminado de la SDDB cuando la aplicación servidora manda un close al *notifier*.

Modos de funcionamiento:

- Modo conectable: un dispositivo en este modo escucha periódicamente intentos de iniciar una conexión de un dispositivo remoto.
- Modo no-conectable: un dispositivo en este modo no escucha intentos de iniciar una conexión de un dispositivo remoto.

Para el correcto funcionamiento de una aplicación servidora, es necesario que el dispositivo servidor esté en modo conectable. Es por esto, que en la implementación de acceptAndOpen(), ésta debe asegurarse que el dispositivo local esté en modo conectable (dado que depende del usuario el tener o no el dispositivo en un modo u otro). La implementación hace una petición al BCC para hacer al dispositivo local conectable, si ésta no es posible, se lanzará una excepción BluetoothStateException.

Cuando todos los servicios en la SDDB han sido eliminados o deshabilitados, la implementación puede decidir opcionalmente pedir al dispositivo servidor que pase al modo no-conectable.

Aunque un dispositivo esté en el modo no-conectable (no responde a intentos de conexión), puede iniciar un intento de conexión. Por esto, un dispositivo en modo no-conectable puede ser un cliente, pero no un servidor. Por lo tanto la implementación no necesita pedir al dispositivo que se ponga en modo conectable si no tiene ningún ServiceRecord en su SDDB.

Clases del Service Registration:

interfaz javax.bluetooth.ServiceRecord: Un ServiceRecord describe un servicio Bluetooth a los clientes. Está compuesto de unos "cuantos" atributos de servicio. El SDP del servidor mantiene una base de datos de los ServiceRecords. Un servicio *run-before-connect* (la aplicación se ejecuta sin esperar a que haya una conexión establecida) añade su ServiceRecord a la SDDB llamando a acceptAndOpen(). El ServiceRecord provee suficiente información a un cliente SDP para poder conectarse al servicio Bluetooth del dispositivo servidor.

La aplicación servidora puede usar el método setDeviceClasses() para activar alguno de los bits de la clase servidora para reflejar la incorporación de un nuevo servicio.

class javax.bluetooth.LocalDevice: Esta clase provee un método getRecord() que la aplicación servidora puede usar para obtener el ServiceRecord. Una vez modificado, puede ser puesto en la SDDB llamando al método notifier.acceptAndOpen() o updateRecord() de LocalDevice.

class javax.bluetooth.ServiceRegistrationException extends java.io.IOException: La excepción ServiceRegistrationException se lanza cuando se intenta añadir o modificar un ServiceRecord en la SDDB y hay algún error. Estos errores pueden ocurrir:

- Durante la ejecución de Connector.open().
- Cuando un servicio *run-before-connect* invoca a acceptAndOpen() y la implementación intenta añadir el ServiceRecord asociado al *notifier* en la SDDB.

• Después de la creación del ServiceRecord, cuando la aplicación servidora intenta modificar el ServiceRecord en la SDDB usando updateRecord().

3.3.3.5 Codificación de una aplicación que haga uso de la API de Bluetooth

A continuación se va a desarrollar un ejemplo que permita descubrir dispositivos (device discovery), y, una vez descubiertos, descubrir qué servicios ofrecen dichos dispositivos (service discovery).

Clase DiscoveryMIDlet

```
package discoveryBluetooth;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import javax.bluetooth.*;
import java.util.*;

public class DiscoveryMIDlet extends MIDlet implements
CommandListener{

//Creamos las variables necesarias
public static DiscoveryMIDlet dm;//Instancia de nuestro objeto
private static Display display;

//Objetos que representan a los dispositivos y a los servicios
private Dispositivo dispositivo = null;
private Servicio servicio = null;

//Objetos Bluetooth necesarios
public LocalDevice dispositivoLocal;
public DiscoveryAgent da;

//Lista de servicios y dispositivos
public static Vector dispositivos_encontrados = new Vector();
public static Vector servicios_encontrados = new Vector();

//Instancia del dispositivo remoto seleccionado
public static int dispositivo_seleccionado = -1;

//-1 indica ninguno seleccionado

//Constructor
public DiscoveryMIDlet(){
dm = this;
}

//Ciclo de vida del MIDlet
public void startApp() {

 display = Display.getDisplay(this);
 dispositivo = new Dispositivo();
 servicio = new Servicio();

 //Mostramos la lista de dispositivos (vacia al principio)
 dispositivo.mostrarDispositivos();
 display.setCurrent(dispositivo);
}
```

```
public void pauseApp() {
}

public void destroyApp(boolean unconditional) {
}

//Este metodo se encarga de las tareas necesarias para salir del
//MIDlet
public static void salir(){

 dm.destroyApp(true);
 dm.notifyDestroyed();
 dm = null;
}

//Este metodo se encarga de dar un aviso de alarma cuando se produce
//una excepcion
public void mostrarAlarma(Exception e, Screen s, int tipo){

 Alert alerta;

      if(tipo == 0){
       alerta = new Alert("Excepcion","Se ha producido la excepcion
       "+e.getClass().getName(), null,
      AlertType.ERROR);
      }

      else{
       alerta = new Alert("Error","No ha seleccionado un dispositivo
       ", null, AlertType.ERROR);
      }

alerta.setTimeout(Alert.FOREVER);
display.setCurrent(alerta,s);
}

//Este metodo se encarga de buscar dispositivos remotos Bluetooth
public void buscarDispositivos(){

 try{
      dispositivoLocal = LocalDevice.getLocalDevice();

      //Ponemos el dispositivo en modo General Discoverable Mode
      //General/Unlimited Inquiry Access Code (GIAC)
      dispositivoLocal.setDiscoverable(DiscoveryAgent.GIAC);
      da = dispositivoLocal.getDiscoveryAgent();
      da.startInquiry(DiscoveryAgent.GIAC,new Listener());
 }

 catch(BluetoothStateException e){
 mostrarAlarma(e, dispositivo,0);
 }
}

//Este metodo se encarga de buscar servicios en un dispositivo
//remoto
public void buscarServicios(RemoteDevice dispositivo_remoto){

 try{
```

```
//Los servicios posibles vienen identificados por un UUID
       int[] servicios = new
       int[]{0x0001,0x0003,0x0008,0x000C,0x0100,0x000F,
       0x1101,0x1000,0x1001,0x1002,0x1115,0x1116,0x1117};
       UUID[] uuid = new UUID[]{new UUID(0x0100)};//Servicios L2CAP
       da.searchServices(servicios,uuid,dispositivo_remoto,new
       Listener());
}

catch(BluetoothStateException e){
 mostrarAlarma(e, dispositivo,0);
}

}

//Manejamos la acción del usuario
public void commandAction(Command c, Displayable d){
if(d==dispositivo && c.getLabel().equals("Descubrir dispositivos")){

      //Limpiamos la lista anterior
      dispositivos_encontrados.removeAllElements();
      //Buscamos dispositivos
      buscarDispositivos();
      //Escribimos un mensaje al usuario pidiendole que espere
      dispositivo.escribirMensaje("Por favor, espere...");
}

else if(d==dispositivo && c.getLabel().equals("Descubrir
Servicios")){

      //Limpiamos la lista anterior
      servicios_encontrados.removeAllElements();
      //Leemos el dispositivo seleccionado
      dispositivo_seleccionado = dispositivo.getSelectedIndex();
      //Nos aseguramos que hay un dispositivo seleccionado

      if(dispositivo_seleccionado == -1) {
       mostrarAlarma(null, dispositivo,1);
       return;
      }

      RemoteDevice dispostivoRemoto =
      (RemoteDevice)dispositivos_encontrados.elementAt(dispositivo_s
      eleccionado);
      buscarServicios(dispostivoRemoto);
}
else if(d==dispositivo && c.getLabel().equals("Salir")){
 salir();
}

else if(d==servicio && c.getLabel().equals("Atras")){
 display.setCurrent(dispositivo);
}

}
```

Clase Listener (se ha incluido dentro de la clase anterior)

```
//Implementamos el DiscoveryListener
public class Listener implements DiscoveryListener{

//Implementamos los metodos del interfaz DiscoveryListener
public void deviceDiscovered(RemoteDevice dispositivoRemoto,
DeviceClass clase){

System.out.println("Se ha encontrado un dspositivo Bluetooth");
dispositivos_encontrados.addElement(dispositivoRemoto);
}

public void inquiryCompleted(int completado){

System.out.println("Se ha completado la busqueda de servicios");

if(dispositivos_encontrados.size()==0){
      Alert alerta = new Alert("Problema","No se ha encontrado
      dispositivos",null, AlertType.INFO);
      alerta.setTimeout(3000);
      dispositivo.escribirMensaje("Presione descubrir
      dispositivos");
      display.setCurrent(alerta,dispositivo);
}

else{
      dispositivo.mostrarDispositivos();
      display.setCurrent(dispositivo);
      }
}

public void servicesDiscovered(int transID, ServiceRecord[]
servRecord){

 System.out.println("Se ha encontrado un servicio remoto");

 for(int i=0;i<servRecord.length;i++){
      ServiceRecord record = servRecord[i];
      servicios_encontrados.addElement(servRecord);
}
}

public void serviceSearchCompleted(int transID, int respCode){

 System.out.println("Terminada la busqueda de servicios");

 if(respCode==SERVICE_SEARCH_COMPLETED ||
      respCode==SERVICE_SEARCH_NO_RECORDS){
      servicio.mostrarServicios();
      display.setCurrent(servicio);
 }
 else{
      Alert alerta = new Alert("Problema","No se ha completado la
      busqueda",null, AlertType.INFO);
      alerta.setTimeout(Alert.FOREVER);
      display.setCurrent(alerta,dispositivo);
      }
 }
}
}
```

Clase Dispositivo:

```
package discoveryBluetooth;
import javax.microedition.lcdui.*;
import javax.bluetooth.*;

public class Dispositivo extends List{

//Constructor
public Dispositivo(){

      super("Lista de dispositivos",List.IMPLICIT);

      addCommand(new Command("Descubrir
      dispositivos",Command.SCREEN,1));
      addCommand(new Command("Descubrir
      servicios",Command.SCREEN,2));
      addCommand(new Command("Salir",Command.SCREEN,3));
      this.setCommandListener(DiscoveryMIDlet.dm);
}

//Este metodo se encarga de limpiar la pantalla
public void limpiar(){

 int s = this.size();

 for(int i=0;i<0;i++){
       delete(i);
 }
}

//Este metodo se encarga de mostrar mensajes
public void escribirMensaje(String str){

 limpiar();
 append(str,null);
}

//Este metodo muestra los "friendly names" de los dispositivos
//remotos
public void mostrarDispositivos(){

limpiar();
if(DiscoveryMIDlet.dispositivos_encontrados.size()>0){

 for(int i=0;i<DiscoveryMIDlet.dispositivos_encontrados.size();i++){

 try{
            RemoteDevice dispositivoRemoto = (RemoteDevice)
            DiscoveryMIDlet.dispositivos_encontrados.elementAt(i);
            append(dispositivoRemoto.getFriendlyName(false),null);
}
 catch(Exception e){
       System.out.println("Se ha producido una excepcion");
 }
}
}

else append("Pulse descubrir dispositivos",null);
}
}
```

Clase Servicio:

```
package discoveryBluetooth;

import javax.microedition.lcdui.*;
import javax.bluetooth.*;
import java.util.*;

public class Servicio extends List{

//Constructor
public Servicio(){

 super("Lista de servicios",List.IMPLICIT);

 addCommand(new Command("Atras",Command.BACK,2));
 this.setCommandListener(DiscoveryMIDlet.dm);
}

//Mostramos la lista de servicios provinientes del record
public void mostrarServicios(){

 int s = this.size();
 for(int i=0;i<s;i++) delete(0);
   for(int j=0;j<DiscoveryMIDlet.servicios_encontrados.size();j++){

   try{

        ServiceRecord rec =
 (ServiceRecord)DiscoveryMIDlet.servicios_encontrados.elementAt(j);

             DataElement e = rec.getAttributeValue(0x0001);
             Enumeration enum = (Enumeration)e.getValue();
             DataElement e2 = (DataElement)enum.nextElement();
             Object v = e2.getValue();
             String name = "#"+j+" "+uuidToName((UUID) v)9;
             append(name, null);
 }

 catch(Exception e){

       System.out.println("Se ha producido una excepcion");
 }
}
}
```

Como se puede observar, esta aplicación en si solo encuentra los dispositivos que haya a su alrededor y extrae los servicios que pueden proporcionar. La API para Bluetooth es más extensa, incorpora clases para el dispositivo (p.e. class javax.bluetooth.LocalDevice o class javax.bluetooth.RemoteDevice) que son utilizadas para acceder y controlar tanto el dispositivo local como remoto. Clases para Seguridad en Bluetooth (estas están incluidas dentro de class javax.bluetooth.RemoteDevice) y tiene métodos para encriptación o autenticación entre otros. Y por supuesto incorpora clases puras de comunicación para realizar la transferencia de archivos entre dispositivos utilizando bluetooth.

A continuación se incluyen algunas imágenes con la emulación de la aplicación.

Figura 97. Menú principal de la aplicación

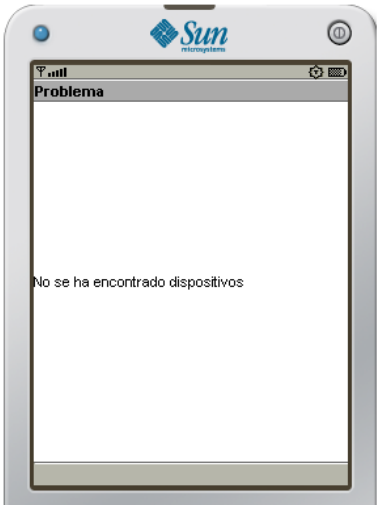


Figura 98. Submenú Descubrir Dispositivos 1

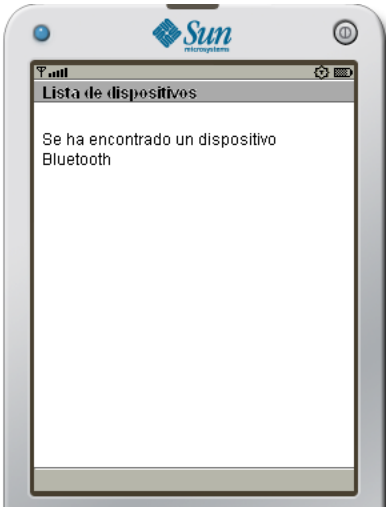


Figura 99. Submenú Descubrir Dispositivos 2

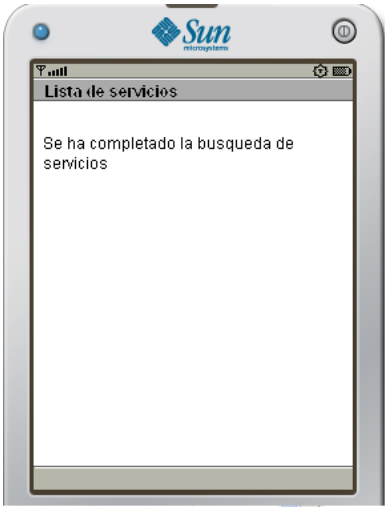


Figura 100. Submenú Descubrir Servicios

Es importante destacar el protocolo que utiliza, este se llama OBEX. En realidad, este, aparentemente no se debería tratar, ya que dicho protocolo OBEX no está definido en el API Bluetooth, si no que tiene su propio API.

Este es un protocolo diseñado por el IrDA (Infrared Data Association) para intercambiar objetos entre clientes y servidores, mediante el establecimiento de una sesión OBEX. En vez de incluir esta funcionalidad en el API Bluetooth, se ha optado por extender el API OBEX y dar soporte a Bluetooth.

El paquete completo javax.obex, se compone de las siguientes interfaces y clases:

Package javax.obex

OBEX (Object Exchange Protocol) classes and interfaces required by JSR 82.

Interface Summary	
Authenticator	This interface is defined by the JSR-82 specification *Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.*
ClientSession	This interface is defined by the JSR-82 specification *Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.*
HeaderSet	This interface is defined by the JSR-82 specification *Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.*
Operation	This interface is defined by the JSR-82 specification *Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.*
SessionNotifier	This interface is defined by the JSR-82 specification *Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.*

Class Summary	
PasswordAuthentication	This class is defined by the JSR-82 specification *Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.*
ResponseCodes	This class is defined by the JSR-82 specification *Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.*
ServerRequestHandler	This class is defined by the JSR-82 specification *Java™ APIs for Bluetooth™ Wireless Technology, Version 1.1.*

Para acceder directamente a esta API sobre la que se estaba hablando de J2ME en web esta el enlace:

http://java.sun.com/javame/reference/apis/jsr082

En ella como se puede comprobar se pueden ver y estudiar a fondo cada uno de los métodos que componen esta clases e interfaces.

Con esta última explicación se quiere resaltar las múltiples posibilidades que proporciona Bluetooth y que lo aquí explicado es una iniciación que es muy fácilmente ampliable con un repaso de la API completa.

3.3.4 Principales APIs de J2ME

3.3.4.1 Introducción

La mayoría de las librerías incluidas en J2ME son un subconjunto de las incluidas en las ediciones J2SE y J2EE de Java. Esto es así para asegurar la compatibilidad y portabilidad de aplicaciones. El mantenimiento de la compatibilidad es un objetivo muy deseable. Las librerías incluidas en J2SE y J2EE tienen fuertes dependencias internas que hacen que la construcción de un subconjunto de ellas sea muy difícil en áreas como la seguridad, E/S, interfaz de usuario, trabajo en red y almacenamiento de datos. Desafortunadamente, estas dependencias de las que hemos hablado hacen muy difícil tomar partes de una librería sin incluir otras. Por esta razón, se han rediseñado algunas librerías, especialmente en las áreas de trabajo en red y Entrada/Salida.

Las librerías de J2ME pueden ser divididas en dos categorías:

- Clases que son un subconjunto de las librerías de J2SE.
- Clases específicas de J2ME.

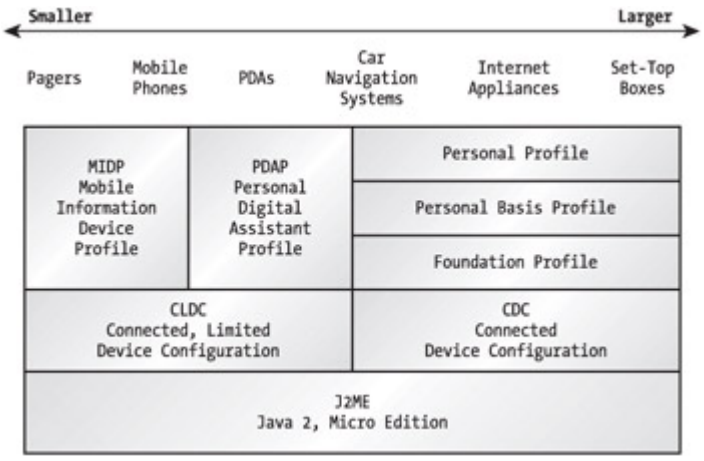


Figura 101. Diagrama de las funcionalidades y recursos de J2ME

En este apartado, a parte de establecer en tablas todas las clases que componen el API de J2ME, se centrara en la explicación de las Interfaces de usuario y todos los elementos que las componen, ya que estas serán las encargadas de facilitar al programador un entorno manejable para crear aplicaciones. El uso de estas interfaces es prioritario en J2ME ya que son las que permiten interactuar con las aplicaciones, por eso se le ha querido dar un énfasis especial en este proyecto a las mismas y dedicar este capitulo a una explicación minuciosa de sus clases y manejo de los eventos que producen.

Teniendo en cuenta la diversidad de aplicaciones que se pueden realizar para los dispositivos MID y los elementos que proporcionan la configuración CLDC y el perfil MIDP, se dividirá a estos elementos en dos grupos:

- Por un lado se verán los elementos que componen la interfaz de usuario de alto nivel. Esta interfaz usa componentes tales como botones, cajas de texto, formularios, etc. Estos elementos son implementados por cada dispositivo y la finalidad de usar las APIs de alto nivel es su portabilidad. Al usar estos elementos, se pierde algo de control del aspecto de la aplicación ya que la estética de estos componentes depende exclusivamente del dispositivo donde se ejecute. En cambio, usando estas APIs de alto nivel se gana un alto grado de portabilidad de la misma aplicación entre distintos dispositivos. Fundamentalmente, se usan estas APIs cuando se quieren construir aplicaciones de negocios o que vayan a ser usadas en diferentes dispositivos.

- Por otro lado están las interfaces de usuario de bajo nivel. Al crear una aplicación usando las APIs de bajo nivel, se tiene un control total de lo que aparecerá por pantalla. Estas APIs proporcionan un control completo sobre los recursos del dispositivo y permiten controlar eventos de bajo nivel como, por ejemplo, el rastreo de pulsaciones de teclas. Generalmente, estas APIs se utilizan para la creación de juegos donde el control sobre lo que aparece por pantalla y las acciones del usuario juegan un papel fundamental.

3.3.4.2 Clases heredadas de J2SE.

J2ME proporciona un conjunto de clases heredadas de la plataforma J2SE. En total, usa unas 37 clases provenientes de los paquetes java.lang, java.util y java.io.

Cada una de estas clases debe ser idéntica o ser un subconjunto de la correspondiente clase de J2SE. Tanto los métodos como la semántica de cada clase deben permanecer invariables. En las siguientes tablas se incluyen todas las clases pertenecientes a J2ME heredadas de J2SE agrupadas por paquetes, según su funcionalidad.

Clases de sistema (Subconjunto de java.lang)
java.lang.Class
java.lang.Object
java.lang.Runnable
java.lang.Runtime
java.lang.String
java.lang.Stringbuffer
java.lang.System
java.lang.Thread
java.lang.Throwable

Clases de Datos (Subconjunto de java.lang)
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.Integer
java.lang.Long
java.lang.Short

Clases de Utilidades (Subconjunto de java.util)
java.util.Calendar
java.util.Date
java.util.Enumeration
java.util.Hashtable
java.util.Random
java.util.Stack
java.util.TimeZone
java.util.Vector

Clases de E/S (Subconjunto de java.io)
java.io.ByteArrayInputStream
java.io.ByteArrayOutputStream
java.io.DataInput
java.io.DataOutput
java.io.DataInputStream
java.io.DataOutputStream
java.io.InputStream
java.io.InputStreamReader
java.io.OutputStream
java.io.OutputStreamWriter
java.io.PrintStream
java.io.Reader
java.io.Writer

3.3.4.3 Clases propias de J2ME

La plataforma J2SE contiene a los paquetes java.io y java.net encargados de las operaciones de E/S. Debido a las limitaciones de memoria de CLDC no es posible incluir dentro de él a todas las clases de estos paquetes.

Ya hemos visto que J2ME hereda algunas clases del paquete java.io, pero no hereda ninguna clase relacionada con la E/S de ficheros, por ejemplo. Esto es debido a la gran variedad de dispositivos que abarca CLDC, ya que, para éstos puede resultar innecesario manejar ficheros.

No se han incluido tampoco las clases del paquete java.net, basado en comunicaciones TCP/IP ya que los dispositivos CLDC no tienen por qué basarse en este protocolo de comunicación.

Para suplir estas "carencias" CLDC posee un conjunto de clases más genérico para la E/S y la conexión en red que recibe el nombre de "*Generic Connection Framework*". Estas clases están incluidas en el paquete javax.microedition.io y son las que aparecen en la siguiente tabla.

Clase	Descripción
Connector	Clase genérica que puede crear cualquier tipo de conexión.
Connection	Interfaz que define el tipo de conexión más genérica.
InputConnection	Interfaz que define una conexión de *streams* de entrada.
OutputConnection	Interfaz que define una conexión de *streams* de salida.
StreamConnection	Interfaz que define una conexión basada en *streams.*
ContentConnection	Extensión a StreamConnection para trabajar con datos.
Datagram	Interfaz genérico de datagramas.
DatagramConnection	Interfaz que define una conexión basada en datagramas.
StreamConnectionNotifier	Interfaz que notifica una conexión. Permite crear una conexión en el lado del servidor.

Estas clases e interfaces se verán con más detenimiento en el último apartado de este capitulo (3.3.5) en el cual se explicarán estas librerías y se creará una aplicación utilizando estas y el resto de las mismas. Aquí sólo se han descrito ya que pertenecen al conjunto de librerías de J2ME y de esto trata el capitulo. En el apartado 3.3.5 se verán en profundidad cada una de estas clases específicas y estudiaremos los métodos que proporcionan.

3.3.4.4 Interfaz de Usuario

Antes de empezar a trabajar con el interfaz de usuario, hay ciertos conceptos que se hacen necesarios comentar para una mejor evolución en el desarrollo. A partir de este momento la "Interfaz de usuario" será referida como UI. Para una referencia mejor a los métodos de cada componente es necesario mirar la especificación de J2ME, que viene generalmente con el SDK, o en java.sun.com.

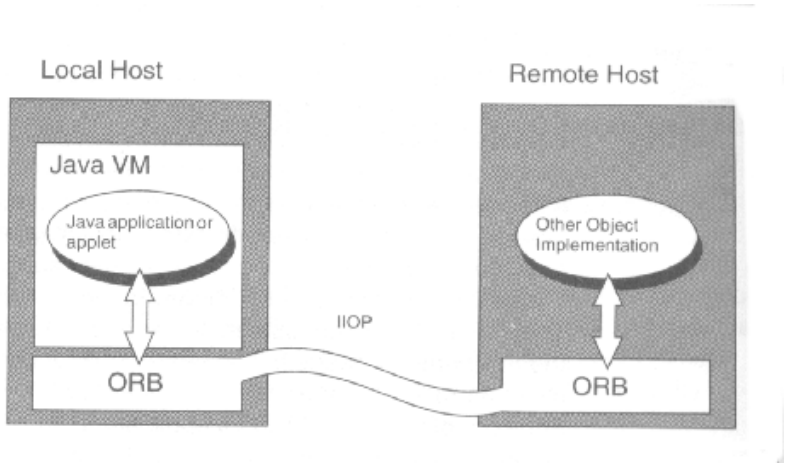


Figura 102. Representación del nivel de interfaz de usuario

En este apartado se explicaran las UI y se irán acompañando de pequeñas aplicaciones que permitan una comprensión de sus distintos aspectos mas real y completa.

El UI de MIDP tiene APIs de alto nivel y de bajo nivel. Las clases de la API de alto nivel, como Alert, Form, List… son extensiones de la clase abstracta Screen. La API de bajo nivel se basa en la clase abstracta Canvas.

La API de alto nivel presenta clases que permite un alto grado de portabilidad entre los diferentes aparatos que soportan J2ME.

La API de bajo nivel permite una manipulación más directa y por ello, mayor control del UI, ya que implementa varios eventos de la pulsación de las teclas y/o pantalla táctil. En este caso se pueden producir errores de portabilidad si los aparatos no tienen las mismas teclas, por ejemplo.

3.3.4.5 Display

Una aplicación puede tener un cierto número de displays, pero solo uno de ellos puede ser visible. Todo MIDlet debe tener al menos un objeto Display. En cualquier caso es muy fácil cambiar el display que se esta mostrando con el comando:

Display.getDisplay(this).setCurrent(nombre_de_Display);

La clase Display implementa la interfaz Displayable con los métodos getCurrent() para saber que display esta en pantalla en ese momento y setCurrent() para cambiar el Display.

Existen 3 tipos de displays:

- Genérico (Form) donde se pueden poner diferentes clases de alto nivel, texto, imágenes…
- Un componente de alto nivel ocupa todo el display (List, TextBox…)
- Bajo nivel display, subclase de la clase Canvas

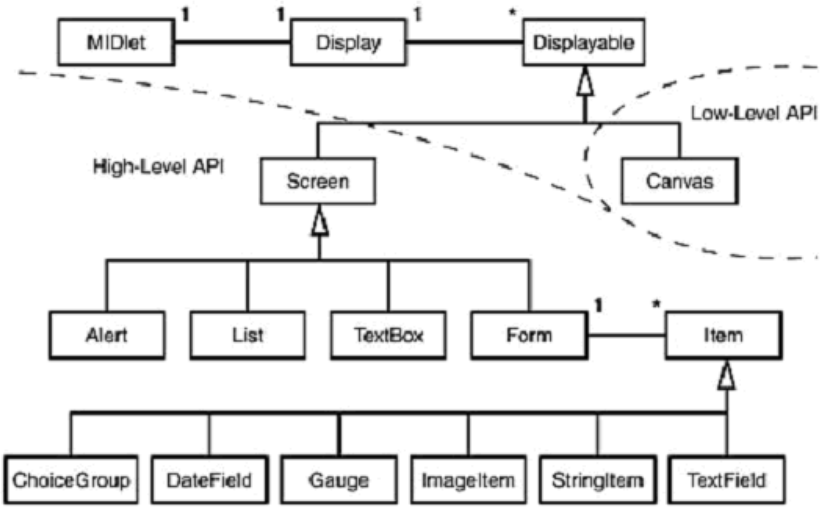


Figura 103. Diagrama con los tipos de Pantallas (displays)

En el caso de que se entre por primera vez en el método startApp, el valor de *pantalla* después de la llamada al método getDisplay() será *null*. Si no es así, es porque se vuelve del estado de pausa por lo que se debe dejar la pantalla tal como está.

Como se ha dicho, cada objeto Display puede tener tantos objetos Displayable como quiera. Como se verá más adelante, las aplicaciones estarán formadas por varias pantallas que serán creadas dentro del método constructor.

La clase abstracta Displayable incluye los métodos encargados de manejar los eventos de pantalla y añadir o eliminar comandos. La siguiente tabla contiene todos los métodos de esta clase:

Métodos	Descripción
void addComand(Command cmd)	Añade el Command *cmd.*
int getHeight()	Devuelve el alto de la pantalla.
Ticker getTicker()	Devuelve el T*icker* (cadena de texto que se desplaza) asignado a la pantalla.
String getTitle()	Devuelve el título de la pantalla.
int getWidth()	Devuelve el ancho de la pantalla.
bolean isShown()	Devuelve *true* si la pantalla está activa.
void removeCommand(Command cmd)	Elimina el Command *cmd*.
void setCommandListener(CommandListener1)	Establece un *listener* para capturar eventos.
void setTicker(Ticker ticker)	Establece un *Ticker* a la pantalla.
void setTitle(String s)	Establece un título a la pantalla.
protected void sizeChanged(int w, int h)	El AMS lo llama cuándo el área disponible para el objeto Displayable es modificada.

A continuación esta el código de una pequeña aplicación que hace uso de los displays y unas imágenes con su emulación en NetBeans:

```
package DisplayTest;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class DisplayTest extends MIDlet implements CommandListener {

    private Command exitCommand;
    private Command changeCommand1;
    private Command changeCommand2;
    private Display display;
    private TextBox t = null;
    private TextBox x = null;

    public DisplayTest() {

       display = Display.getDisplay(this);
       exitCommand = new Command("Salir", Command.EXIT, 2);

       changeCommand1= new Command("Cambiar", Command.SCREEN, 1);
       changeCommand2= new Command("Cambiar", Command.SCREEN, 1);
    }

   public void startApp() {

       t = new TextBox("Hello MIDlet", "Esta es la pantalla número
       1", 256, 0);
       t.addCommand(exitCommand);
       t.addCommand(changeCommand1);

       x = new TextBox("Hello MIDlet", "Esta es la pantalla número
       2", 256, 0);
       x.addCommand(exitCommand);
       x.addCommand(changeCommand2);

       t.setCommandListener(this);
       x.setCommandListener(this);
       display.setCurrent(t);
       }

    public void pauseApp() {//No es necesario implementarlo}

    public void destroyApp(boolean unconditional) {
       //No es necesario implementarlo}

    public void commandAction(Command c, Displayable s) {
       if (c == exitCommand) {
             destroyApp(false);
             notifyDestroyed();
       }
       if (c == changeCommand1) {
             display.setCurrent(x);
       }
       if (c == changeCommand2) {
             display.setCurrent(t);
       }
   }
}
```

A continuación dos capturas de pantalla de el funcionamiento de la aplicación.

Figura 104. Captura de pantalla del emulador de Netbeans con la pantalla número 1.

Figura 105. Captura de pantalla del emulador de Netbeans con la pantalla número 2.

3.3.4.6 Comandos

Toda interacción entre el usuario y el programa se realiza a través de los comandos. Estos funcionan de forma parecida que J2SE., mediante un CommandListener que recibe el comando y la pantalla de donde viene. Para implementar un comando se usa el método addCommand().

Ejemplo:

Command OK_CMD = new Command("Ok", Command.OK, 1);
myDisplay.addCommand(OK_CMD);

Existen tres parámetros que hay que definir cuando construimos un objeto Command:

• Etiqueta: La etiqueta es la cadena de texto que aparecerá en la pantalla del dispositivo que identificará a nuestro Command.
• Tipo: Indica el tipo de objeto Command que queremos crear. La declaración del tipo sirve para que el dispositivo identifique el Command y le dé una apariencia específica acorde con el resto de aplicaciones existentes en el dispositivo.
• Prioridad: Es posible asignar una prioridad específica a un objeto Command. Esto puede servirle al AMS para establecer un orden de aparición de los Command en pantalla. A mayor número, menor prioridad.

Tipos de comando:

- BACK: Petición para volver a la pantalla anterior.
- CANCEL: Petición para cancelar la acción en curso
- EXIT: Petición para salir de la aplicación
- HELP: Petición para mostrar información de ayuda
- ITEM: Petición para introducir el comando en un "item" en la pantalla
- OK: Aceptación de una acción por parte del usuario
- SCREEN: Para Commands de propósito más general
- STOP: Petición para parar una operación

Los métodos de la Clase Command son los siguientes:

Métodos	Descripción
public int getCommandType()	Devuelve el tipo del Command.
public String getLabel()	Devuelva la etiqueta del Command.
public String getLongLabel()	Devuelve la etiqueta larga del Command.
public int getPriority()	Devuelve la prioridad del Command.

A continuación se muestra el código una aplicación que hace uso de comandos (en esta caso se usa el comando Salir), de un Display y de un Form (el cual se verá a continuación). Esta aplicación muestra los recursos y propiedades de un Midlet. Se ha implementando el commandAction e incorpora unas imágenes de su emulación con NetBeans:

```
package MIDletProps;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


public class MIDletProps extends MIDlet implements CommandListener{

    private Display display;    // Este es el Display para el MIDlet
    private Form props;
    private StringBuffer propbuf;
    private Command exitCommand = new Command("Salir",
 Command.SCREEN, 1);

    public MIDletProps() {
        display = Display.getDisplay(this);
    }

    public void startApp() {
       Runtime runtime = Runtime.getRuntime();
       runtime.gc();
       long free = runtime.freeMemory();
       long total = runtime.totalMemory();

       propbuf = new StringBuffer( 50 );
       props = new Form( "Propiedades del Sistema" );

       props.append( "Memoria Libre = " + free + "\n" );
       props.append( "Memoria Total = " + total + "\n" );

       props.append( showProp( "microedition.configuration" ) );
       props.append( showProp( "microedition.platform" ) );
       props.append( showProp( "microedition.locale" ) );
       props.append( showProp( "microedition.encoding" ) );
       props.append( showProp( "microedition.encodingClass" ) );
       props.append( showProp( "microedition.http_proxy" ) );

       props.addCommand( exitCommand );
       props.setCommandListener( this );

       display.setCurrent( props );

    }

    public void commandAction( Command c, Displayable s )
    {
      if ( c == exitCommand )
      {
         destroyApp( false );
         notifyDestroyed();
      }
    }
       String showProp( String prop )
   {
```

```
        String value = System.getProperty( prop );
        propbuf.setLength( 0 );
        propbuf.append( prop );
        propbuf.append( " = " );
        if (value == null)
        {
           propbuf.append( "<undefined>" );
        }
        else
        {
            propbuf.append( "\"" );
            propbuf.append( value );
            propbuf.append( "\"" );
        }
        propbuf.append( "\n" );
        return propbuf.toString();
    }

    public void pauseApp(){
       display.setCurrent( null );
       propbuf = null;
       props = null;
    }


    public void destroyApp( boolean unconditional ){
        //No hace falta implementarlo
    }
}
```

Figura 106. Emulación de la aplicación MIDletProps

APIs de Alto Nivel

3.3.4.7 List

La clase List permite construir pantallas que poseen una lista de opciones. Se utilizará para construir menús de manera independiente. Esta clase implementa a la interfaz Choice, lo cual permitirá al programador construir tres tipos de listas diferentes.

Existen dos constructores que permiten construir listas. El primero construirá una lista vacia (List (String Titulo, int listType)) y el segundo construirá una lista con un conjunto inicial de opciones e imágenes (List (String Titulo, int listType, String[] Elementos, Image[] Imagenes)).

En una lista de opciones es posible añadir, insertar en medio o borrar opciones. Las opciones pueden poseer texto y/o una imagen.

Hay 3 tipos de listas:

- **EXCLUSIVE**: lista en la que solo se puede tener una opción seleccionada.
- **IMPLICIT**: lista en la que la selección de un elemento provoca un evento.
- **MULTIPLE**: lista en la que cualquier número de elementos pueden ser seleccionados.

A continuación se añade el código de una aplicación que crea varias listas, exactamente una de cada tipo (Exclusive, Implicit…) se le añade a una de ellas una imagen. También se añaden unas figuras con la emulación del mismo con Netbeans:

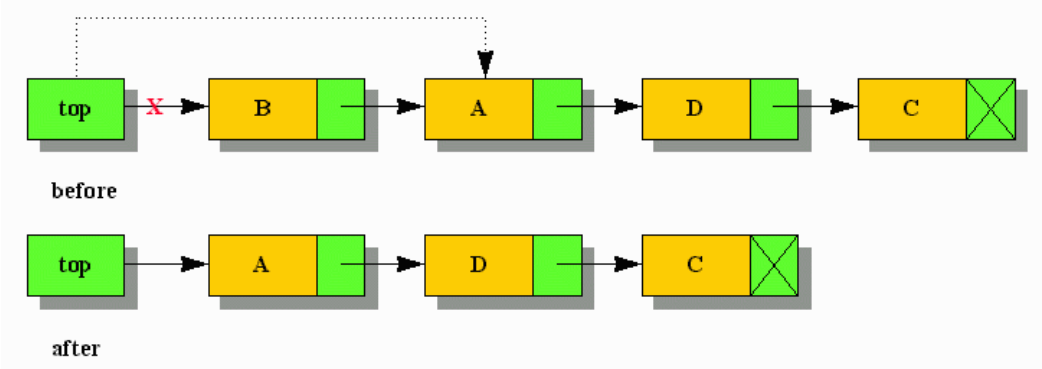


Figura 107. Ilustración del tratamiento de listas, en concreto una lista de la que se elimina un término.

A continuación el código de la aplicación desarrollada:

```
package ListTest;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


public class ListTest extends MIDlet implements CommandListener{
 private List menu;
 private List implicit_list;
 private List exclusive_list;
 private List multiple_list;
 private Command okCommand = new Command("OK", Command.OK, 1);
 private Command backCommand = new Command("Atrás", Command.BACK,
 1);
 private String[] options={"Opción A","Opción B","Opción C"};
 private Display display;

    public ListTest() {

       //Se crea una lista implicita para mostrar un menu
       menu= new List("Elige modo", List.IMPLICIT);

       try {
             menu.append("Lista eleccion
             Implicita",Image.createImage ("/icons/pic.png"));
       }
        catch (java.io.IOException x) {
             throw new RuntimeException ("Imagen no encontrada");
       }

       menu.append("Lista de eleccion multiple",null);
       menu.insert(1, "Lista de eleccion Exclusiva",null);
       menu.append("Salir",null);
       menu.setCommandListener(this);

       //Creación de la lista IMPLICIT
      implicit_list= new List("Lista Eleccion-
IMPLICIT",List.IMPLICIT,options, null);

      //Se añade el Comando
       implicit_list.addCommand(backCommand);
       implicit_list.setCommandListener(this);

       //Creacion de la lista EXCLUSIVE
       exclusive_list= new List("Lista Eleccion-
       EXCLUSIVE",List.EXCLUSIVE, options, null);

       exclusive_list.addCommand(okCommand);
       exclusive_list.addCommand(backCommand);
       exclusive_list.setCommandListener(this);

       //Creacion de lista de Multiple eleccion
       multiple_list= new List("Lista Eleccion-
       Multiple",List.MULTIPLE,options, null);
       multiple_list.addCommand(okCommand);
       multiple_list.addCommand(backCommand);
       multiple_list.setCommandListener(this);
```

```
            display=Display.getDisplay(this);
      }

         public void startApp() throws MIDletStateChangeException {
             display.setCurrent(menu);
         }

         public void pauseApp() {//No se implementa}

         public void destroyApp(boolean unconditional) {
             //Vacio los recursos
            menu=null;
            implicit_list=null;
            exclusive_list=null;
            multiple_list=null;
            okCommand = null;
            backCommand = null;
            display=null;
         }

         public void commandAction(Command c, Displayable d) {
            if(d==menu && c==List.SELECT_COMMAND) {
                  switch(menu.getSelectedIndex()) {
                   case 0: //lista eleccion implicita
                         display.setCurrent(implicit_list);
                         break;
                  case 1: //lista eleccion exclusive
                         display.setCurrent(exclusive_list);
                         break;
                  case 2: //lista eleccion multiple
                         display.setCurrent(multiple_list);
                         break;
                  case 3: //salir
                         destroyApp(true);
                         notifyDestroyed();
                         break;
                  default:
                  }
            }
            else if(d==implicit_list && c==List.SELECT_COMMAND) {
              System.out.println(options[((List)d).getSelectedIndex()]+"
      es seleccionada");
      }
            else if(d==exclusive_list && c==okCommand) {
            System.out.println(options[((List)d).getSelectedIndex()]+" es
seleccionada");
      }
            else if(d==multiple_list && c==okCommand) {
            //Devuelve que opción fue seleccionada
                  boolean[] selected= new boolean[options.length];
                  ((List)d).getSelectedFlags(selected);
                  for(int i=0; i<options.length; i++) {
                   if(selected[i]) {
                   System.out.println(options[i]+" es seleccionada");
            }
          }
      }
      else if(c==backCommand) {
          display.setCurrent(menu);
      }  }}
```

Figura 108.Imagen de Menu principal

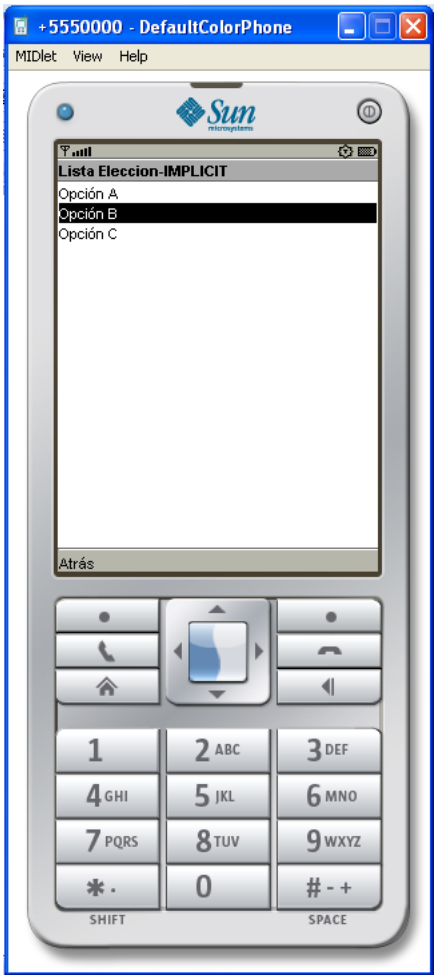


Figura 109. Imagen de selección de primera lista (IMPLICIT)

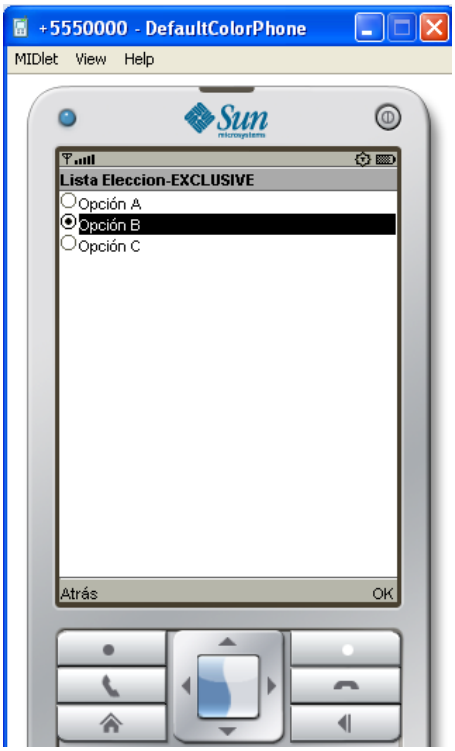


Figura 110. Imagen de selección lista (EXCLUSIVE)

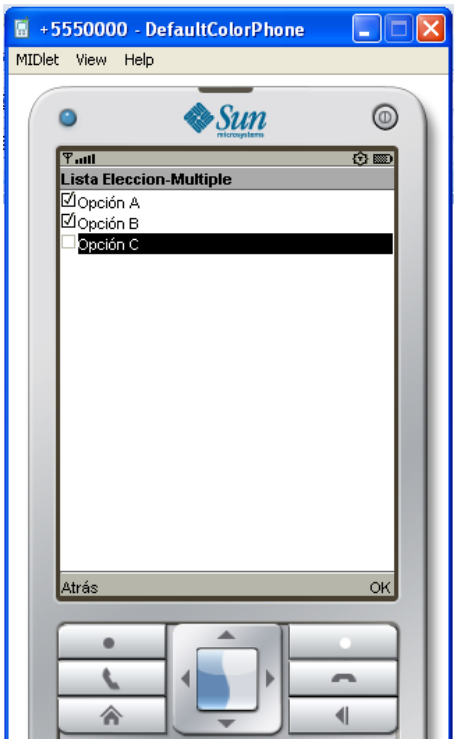


Figura 111. Imagen de selección lista (MULTIPLE)

3.3.4.8 TextBox

Un TextBox es una pantalla que permite insertar texto en ella. Cuando se crea el TextBox hay que especificar el tamaño del mismo, es decir, el número de caracteres que puede albergar. Si la capacidad elegida es mayor que la que puede mostrar el dispositivo a la vez se genera un mecanismo de scroll para la correcta visualización de todo el TextBox.

A un TextBox se le pueden añadir las siguientes restricciones:

- **ANY**: Permite la introducción de cualquier tipo de texto.
- **CONSTRAINT_MASK**: Se usa cuándo necesitamos determinar el valor actual de las restricciones
- **EMAILADDR**: Permite caracteres validos para direcciones de correo electrónico.
- **NUMERIC**: Permite solo números enteros, positivos o negativos.
- **PASSWORD**: Oculta los caracteres introducidos mediante una máscara para proporcionar privacidad.
- **PHONENUMBER**: Permite caracteres validos solo para números de teléfono.
- **URL**: Permite caracteres válidos sólo para direcciones URL.

Algunos métodos a tener en cuenta son, delete(), insert(), setChars(), setString(). Aunque posee bastantes métodos más.

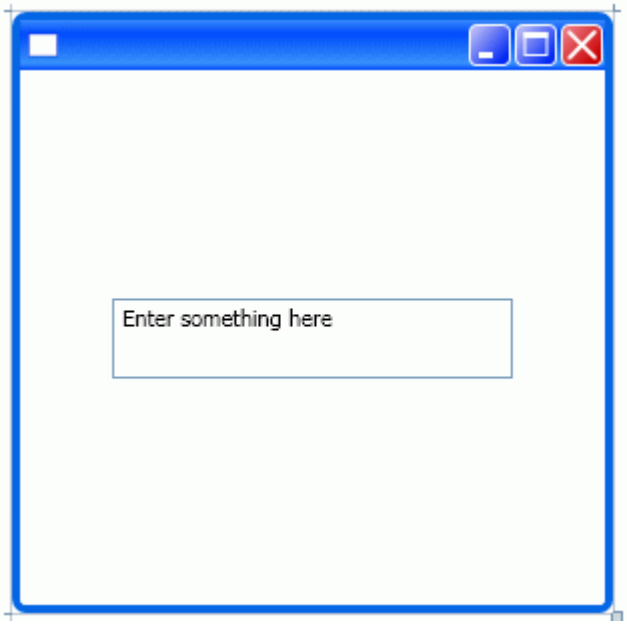


Figura 112. Apariencia de un Textbox.

3.3.4.9 Form

Un formulario (Form) es un componente que actúa como contenedor de un número indeterminado de objetos. Todos los objetos que puede contener Form derivan de la clase Item.

El número de objetos que puede contener Form es variable, pero teniendo en cuenta el tamaño de las pantallas no es aconsejable que el número de objetos introducidos sea grande ya que se producirá un scroll.

Para referirnos a los Items o componentes de un formulario usaremos unos índices, siendo 0 el índice del primer Item y Form.size()-1 el del último. El método size() nos devuelve el número de Items que contiene un formulario. Un mismo Item no puede estar en más de un formulario a la vez. Si, por ejemplo, deseamos usar una misma imagen en más de un formulario, deberemos borrar esa imagen de un formulario antes de insertarla en el que vamos a mostrar por pantalla. Si no cumplimos esta regla, se lanzaría la excepción IllegalStateException.

Para manejar estas excepciones se hará de manera similar al de la clase Commands. Es necesario implementar la interfaz ItemStateListener que contiene un solo método abstracto itemStateChanged(Item item).

Cuando realizamos algún tipo de acción en un Item de un formulario, ejecutamos el código asociado a ese Item que definamos en el método itemStateChanged(Item item) de igual forma que hacíamos con el método commandAction(Command c, Displayable d) cuando manejamos Commands.

Usando la clase Form, podemos tener diferentes elementos dentro de este contenedor, estos pueden ser:

- StringItem, es la clase mas simple que deriva de Item. Es una cadena de texto no modificable.
- ChoiceGroup, es un grupo de elementos que podemos seleccionar.
- DateField, muestra fechas y hora, según que teléfonos incluso puede mostrar un calendario.
- Gauge, es un indicador de progresos a través de un gráfico de barras.
- ImageItem, nos da la posibilidad de incluir imágenes en el form.
- ListBox, una lista de valores
- TextField, como el TextBox anterior, es un campo de texto.

A continuación esta el código de una aplicación que hace uso de Form (FormTest), en esta aplicación lo que se ha hecho ha sido añadir diferentes tipos de elementos que puede contener un Form. El código va acompañado por una figura con la emulación de la aplicación en NetBeans:

```
package FormTest;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FormTest extends MIDlet implements CommandListener{

    private Form    mainscreen;
    private Command okCommand = new Command("OK", Command.OK, 1);
    private Command exitCommand = new Command("Salir", Command.EXIT,
 1);
    private Display display;

    public FormTest () {

        mainscreen= new Form("Demo Form");
        mainscreen.addCommand(okCommand);
        mainscreen.addCommand(exitCommand);
        mainscreen.setCommandListener(this);

        //Creo un StringItem
        mainscreen.append(new StringItem("StringItem:”,"Aquí"));

        //Creo una ImageItem
        Image img=null;
        try {
            img= Image.createImage("/icons/open.png");
        }
       catch (Exception e) {}

        mainscreen.append(new ImageItem("ImageItem:",img,
       ImageItem.LAYOUT_CENTER,"No se puede mostrar imagen"));

        //Creo un Choice
        String[] editable_choices={"Opcion 1", "Opcion 2"};
        mainscreen.append(new ChoiceGroup("Opciones
       Gauge",Choice.EXCLUSIVE,editable_choices,null));

       //Creo un TextField
        mainscreen.append(new TextField("TextField", "", 20,
        TextField.ANY));

        //Creo un DateField
        mainscreen.append(new DateField("DateField",
        DateField.DATE));

        //Creo un Gauge
        mainscreen.append(new Gauge("Gauge:",true,100,50));

        display=Display.getDisplay(this);
    }

     public void startApp() throws MIDletStateChangeException {
        display.setCurrent(mainscreen);
    }

    public void pauseApp() {
        //Innecesaria su implementación
    }
```

```
    public void destroyApp(boolean unconditional) {
       //Vacio los recursos
       mainscreen=null;
       okCommand = null;
       exitCommand = null;
       display=null;
    }

    public void commandAction(Command c, Displayable d) {
      if(c==okCommand) {
      }

      else if(c==exitCommand) {
          destroyApp(true);
          notifyDestroyed();
      }
    }
}
```

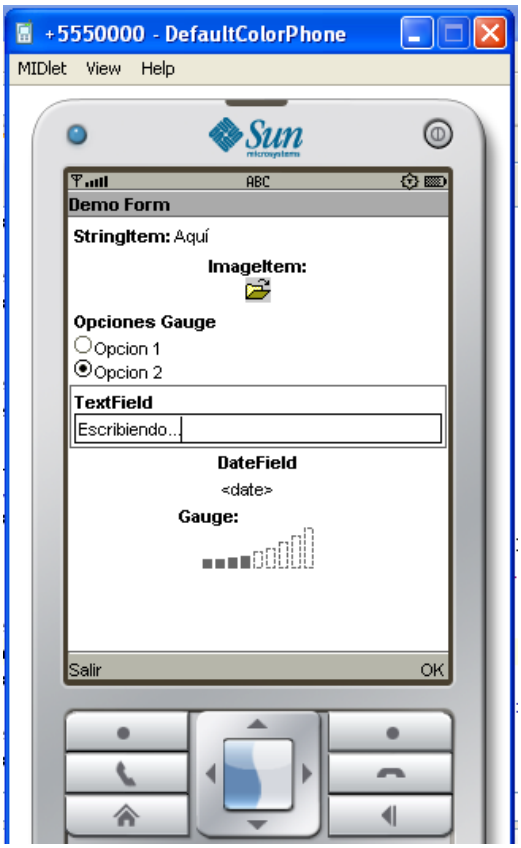


Figura 113. Captura de pantalla de la aplicación FormTest

3.3.4.10 Alerts

El objeto Alert representa una pantalla de aviso. Normalmente se usa cuando se quiere avisar al usuario de una situación especial como, por ejemplo, un error. Un Alert está formado por un título, texto e imágenes si se quiere.

Para crear una pantalla de alerta se cuenta con dos constructores:

Alert(String titulo)
Alert(String titulo, String textoalerta, Image imagen, AlertType tipo)

Además podemos definir el tiempo que queremos que el aviso permanezca en pantalla, diferenciando de esta manera dos tipos de Alert:

1. Modal: La pantalla de aviso permanece un tiempo indeterminado hasta que es cancelada por el usuario. Esto se consigue invocando al método Alert.setTimeOut(Alert.FOREVER).

2. No Modal: La pantalla de aviso permanece un tiempo definido por el usuario. Para ello se indicará el tiempo en el método setTimeOut(tiempo). Una vez finalizado el tiempo, la pantalla de aviso se eliminará de pantalla y aparecerá el objeto Displayable que definamos.

Podemos elegir el tipo de alerta que vamos a mostrar. Cada tipo de alerta tiene asociado un sonido. Los tipos que podemos definir son:

- **ALERT:** Aviso de una petición previa.
- **CONFIRMATION:** Indica la aceptación de una acción.
- **ERROR:** Indica que ha ocurrido un error.
- **INFO:** Indica algún tipo de información.
- **WARNING:** Indica que puede ocurrir algún problema.

Es necesario especificar que display se va a mostrar después de la alerta

Display.getDisplay(this).setCurrent(alerta, display_de _despues);

Es posible ejecutar el sonido sin tener que crear un objeto Alert, invocando al método playSound(Display) de la clase AlertType, por ejemplo:

AlertType.CONFIRMATION.playSound(display)

A continuación una aplicación que hace uso de Alerts (AlertTest), se ha creado un TextBox con una gran capacidad, si se escriben 6 caracteres y se pulsa OK sonará el sonido de la Alarma “INFO” y se mostrará por pantalla un mensaje. Si en caso contrario se escriben un número distinto a 6 caracteres sonará el sonido de Error y se mostrará por pantalla el mensaje de error. A la hora de emular esta aplicación es importante tener los altavoces del ordenador encendidos ya que los sonidos provienen de ellos. El código se acompañará tambié de unas imágenes con la emulación de la aplicación en NetBeans.

Clase AlertTest:

```
package AlertTest;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class AlertTest extends MIDlet implements CommandListener{
    private TextBox textBox;
    private Command okCommand = new Command("OK", Command.OK, 1);
 private Command exitCommand = new Command("EXIT", Command.EXIT, 1);
    private Display display;

    public AlertTest() {
        //Creo una lista exclusiva
       textBox= new TextBox("TextBox de 6
       Caracteres","",20,TextField.ANY);
       textBox.addCommand(okCommand);
       textBox.addCommand(exitCommand);
       textBox.setCommandListener(this);

       display=Display.getDisplay(this);
    }

    public void startApp() throws MIDletStateChangeException {
        display.setCurrent(textBox);
    }

    public void pauseApp() {
        //No hace falta implementarlo
    }

    public void destroyApp(boolean unconditional) {
        //Vacio los recursos
        textBox=null;
       okCommand = null;
       exitCommand = null;
        display=null;
    }

     public void commandAction(Command c, Displayable d) {

       if(d==textBox && c==okCommand) {
            String t=textBox.getString();
            if(t.length()==6) {
             Alert info= new Alert("OK","Todo correcto.Prueba de
             nuevo",null,AlertType.INFO);
             info.setTimeout(Alert.FOREVER);
             display.setCurrent(info,textBox);
            }
       else {
             Image err_img= null;
             //create error image
             try {
                   err_img=Image.createImage("/icons/error.png");
               }
             catch(Exception e){
              System.out.println("La imagen no se ha podido abrir");
                }
```

```
                Alert error= new Alert("Error","Error, Prueba otra
            vez.",err_img,AlertType.ERROR);
                    error.setTimeout(Alert.FOREVER);
                    display.setCurrent(error,textBox);
                }
    }
            else if(c==exitCommand) {
                destroyApp(true);
                notifyDestroyed();
            }
        }
}
```

Figura 114. Imagen de la emulación palabra 6 caractéres

Figura 115. Al pulsar OK, suena la Alerta INFO

Figura 116. Imagen de la emulación palabra 8 caracteres

Figura 117. Al pulsar OK, suena la Alerta ERROR

3.3.4.11 Timers

Permite realizar la ejecución de código en intervalos de tiempo.

- Timer.schedule(TimerTask, Date)
- Timer.schedule(TimerTask, Date, period)
- Timer.schedule(TimerTask, long delay, long period)

A continuación se ha desarrollado un ejemplo con Timer. Para ello se ha creado una aplicación que simula la compra de Stocks en una tienda en tiempo real. El display contiene un Flicker que va marcando el precio de lo que se quiere comprar en cada momento, y un menú en el que se puede agregar Stock o eliminar Stock de un conjunto de marcas de móviles. También se puede elegir en un menú desplegable el tiempo que tardara el Flicker en actualizarse (todo esto usando la clase Timer). A continuación se pondrá el código de la aplicación acompañado de unas figuras con su emulación en Netbeans.

Clase TimerMIDlet:

```
package TimerMIDlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.Timer;              //Añado librerías clase Timer
import java.util.TimerTask;


public class TimerMIDlet extends MIDlet implements CommandListener {

    private Display display = null;

    private Ticker stockTicker = null;
    private String[] stocks = { "SUNW", "ORCL", "NOK", "MOT" };
    private String[] precios = { "105.1", "75", "40.25", "35.5"};

    private static final Command exitCommand = new Command("Salir",
    Command.STOP,0);
    private static final Command backCommand = new Command("Volver",
    Command.BACK,0);
    private static final Command doneCommand = new Command("Hecho",
    Command.OK,0);

    private Timer stockRefresh = null;
    private StockRefreshTask stockRefreshTask = null;

    private int refresh_interval = 1000; // 1000 = 1 segundo

    private List menu = null;
    private ChoiceGroup updatesChoices = null;
    private Form updatesForm  = null;
    private String currentMenu = null;

    public TimerMIDlet() {//Constructor Vacio
    }

   public void startApp() throws MIDletStateChangeException {

        display = Display.getDisplay(this);
```

```
            menu = new List("Menu Stock", Choice.IMPLICIT);
            menu.append("Actualizar", null);
            menu.append("Añadir Stocks", null);
            menu.append("Eliminar Stocks", null);
            menu.addCommand( exitCommand );
            menu.setCommandListener( this );

           // Creo el Ticker
            stockTicker = new Ticker( makeTickerString() );
            menu.setTicker(stockTicker);

            display.setCurrent( menu );
            currentMenu = "Menu Stock";

            updatesForm = new Form("Actualizar");
            updatesChoices = new ChoiceGroup("Intevalo de
           Actualizacion:", Choice.EXCLUSIVE);
            updatesChoices.append("Continuar", null); // 1 segundo
            updatesChoices.append("5 sc", null);
            updatesChoices.append("10 sc", null);
            updatesChoices.append("15 sc", null);
            updatesChoices.append("20 sc", null);

            updatesForm.setTicker(stockTicker);
            updatesForm.append( updatesChoices);
            updatesForm.addCommand(backCommand);
            updatesForm.addCommand(doneCommand);
            updatesForm.setCommandListener( this );

            //Reinicio el Timer para actualizar el stock
            stockRefreshTask = new StockRefreshTask();
            stockRefresh = new Timer();
            stockRefresh.schedule(stockRefreshTask, 0,
           refresh_interval);
        }

        public String makeTickerString()
        {
           String retString = new String();
           for ( int i = 0; i < stocks.length; i++ )
           {
              retString += stocks[i];
              retString += " @ ";
              retString += precios[i];
              retString += " ";
           }
           return retString;
        }

         public void pauseApp() {
            // Vacio los recursos
            display = null;
            stockRefresh = null;
            stockRefreshTask = null;
        }

          public void destroyApp(boolean unconditional) throws
MIDletStateChangeException {
                notifyDestroyed();
        }
```

```
  public void commandAction(Command c, Displayable d)
        {
            if (c == exitCommand)
            {
               try
               {
                  destroyApp(false);

               }
               catch (MIDletStateChangeException msce)
               {
                  System.out.println( "Error en destroyApp(false) " );
                  msce.printStackTrace();
               }
                notifyDestroyed();
            }
            else if ( c == backCommand )
            {
               currentMenu = "Menu Stock";
               display.setCurrent( menu );
            }
            else if ( c == doneCommand )
            {
               switch( updatesChoices.getSelectedIndex() )
               {
                   case 0:
                       refresh_interval =    1000;
                       break;
                   case 1:
                       refresh_interval =   5000;
                       break;
                   case 2:
                       refresh_interval =  10000;
                       break;
                   case 3:
                       refresh_interval =  15000;
                       break;
                   case 4:
                       refresh_interval = 20000;
                       break;
                   default:
                       break;
               }
               stockRefreshTask.cancel();
      stockRefreshTask = new StockRefreshTask();
               stockRefresh.schedule(stockRefreshTask, 0,
refresh_interval);
               display.setCurrent( menu );
               currentMenu = "Menu Stock";

            }
            else
            {
               List shown = ( List )display.getCurrent();
               switch (shown.getSelectedIndex())
               {
                  case 0:  // Actualizar
                     display.setCurrent( updatesForm );
                     currentMenu = "Actualizar";
                     break;
                  case 1:  // Añadir Stock
```

```
                System.out.println( "Añadir Stock... " );
                currentMenu = "Añadir Stock";
                break;
            case 2:  // Eliminar Stock
                System.out.println( "Eliminar Stock... " );
                currentMenu = "Eliminar Stock";
                break;
        }
    }
}
```

A continuación dos imágenes con la emulación de la aplicación en Netbeans.

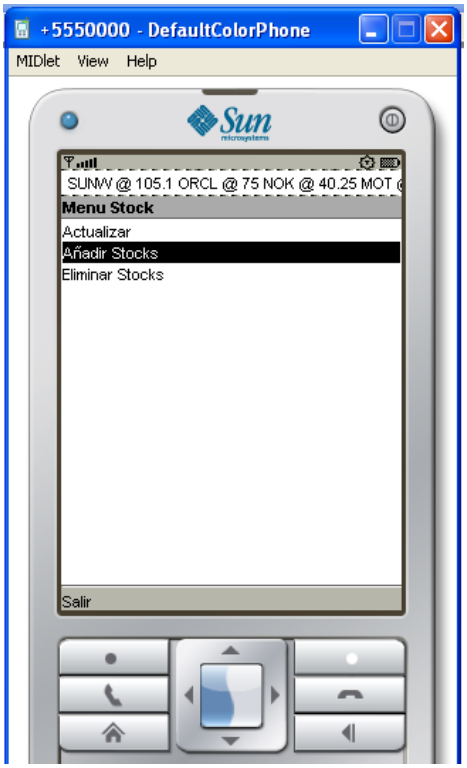


Figura 118. Menú principal de la aplicación

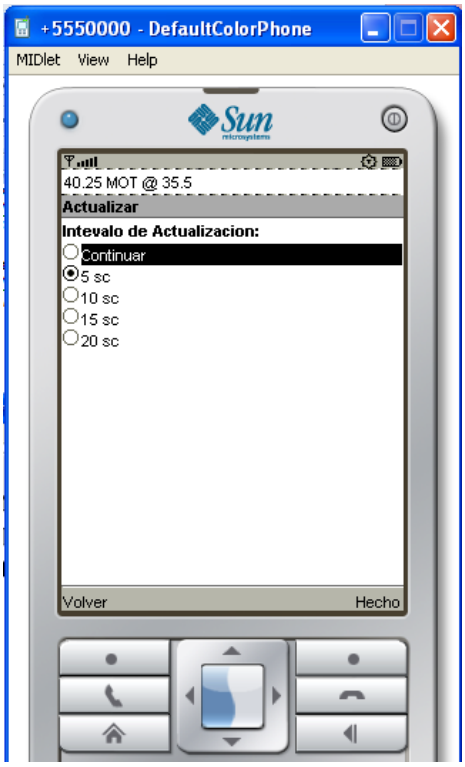


Figura 119. Menú del ChoiceGroup “Actualizar”

APIs bajo nivel

3.3.4.12 Canvas

Para realizar un juego o una aplicación con cierta movilidad, no se pueden usar los componentes antes mencionados. Para ello se hará uso del Canvas. La interfaz Canvas posee un método de implementación obligatoria, paint() que es llamado cuando el Canvas tiene que ser dibujado. La clase Canvas es la superclase de todas las pantallas que usan las APIs de bajo nivel, al igual que Screen lo era para las pantallas que usaban las APIs de alto nivel. No existe ningún impedimento que nos permita usar en el mismo MIDlet pantallas tanto derivadas de Canvas como de Screen.

También posee los métodos getHeigh() y getWidth() para saber el tamaño del canvas y poder adaptar nuestras aplicación a distintos teléfonos.

Algo muy común al empezar el método paint() es limpiar la pantalla para no dibujar sobre el anterior grafico, el siguiente código produce el efecto deseado, si el fondo es blanco:

```
g.setGrayScale (255);
g.fillRect (0, 0, getWidth (), getHeight ());
g.setGrayScale (0);
```

Nokia posee una versión especial, llamada FullCanvas que ocupa toda la pantalla, mientras que Canvas, deja la barra de titulo y otras partes de la aplicación visibles.

Además de paint(), canvas posee otros métodos, como getGameAction(int keyCode) que registra el uso de las teclas por parte del usuario. Otros métodos que canvas facilita al programador cuando suceden ciertos eventos son:

- **showNotify**() cuando se muestra el canvas
- **hideNotify**() cuando se oculta el canvas
- **keyPressed**() para el control de pulsación de teclas
- **keyRepeated**() para el control de pulsación de teclas
- **keyReleased**() para el control de pulsación de teclas
- **pointerPressed**() para el control de la pantalla táctil
- **pointerDragged**() para el control de la pantalla táctil
- **pointerReleased**() para el control de la pantalla táctil

En el método paint, la clase Graphics proporciona los siguientes métodos para dibujar:

- **drawImage** (Image image, int x, int y, int align)
- **drawString** (String text, int x, int y, int align)
- **drawRect** (int x, int y, int w, int h)
- **drawRoundRect** (int x, int y, int w, int h, int r)
- **drawLine** (int x0, int y0, int x1, int y1)
- **drawArc** (int x, int y, int w, int h, int startAng, int arcArc)
- **fillRect** (int x, int y, int w, int h)
- **fillRoundRect** (int x, int y, int w, int h, int startAng, int endAng)
- **fillArc** (int x, int y, int w, int h, int startAng, int endAng)

Para más información sobre esta clase se puede mirar la documentación sobre J2ME.

Como ejemplo se usarán dos pequeñas aplicaciones de manejo de Canvas muy sencillas. La primera de ellas llamada CanvasTest simplemente pintara en pantalla una línea de color gris y un rectángulo de color azul. Dentro de esta clase ira implementada otra clase derivada de Canvas, es necesario para el uso del MIDlet. La segunda aplicación cargará una imagen en el display en diferentes posiciones para mostrar la manera en que se puede jugar con las coordenadas. Las dos aplicaciones van acompañadas de sus respectivas figuras generadas por el emulador de Netbeans.

Clase CanvasTest.

```
package CanvasTest;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class CanvasTest extends MIDlet {

  public void startApp () {
    Display.getDisplay (this).setCurrent (new TestCanvas ());
  }
  public void pauseApp() {}
  public void destroyApp(boolean forced) {}
}

class TestCanvas extends Canvas {
public void paint (Graphics g) {
       g.setGrayScale (255);
       //Limpio la pantalla
       g.fillRect (0, 0, getWidth (), getHeight ());
       g.setGrayScale (0);
       g.drawLine (0, 0, 100, 200);
       g.setColor(0,0,255);
       g.fillRect (20, 30, 30, 20); }
}
```

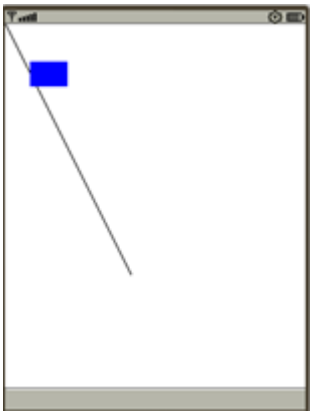


Figura 120. Emulación de la aplicación CanvasTest

Clase ImageTest.

```
package ImageTest;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.io.*;

public class ImageTest extends MIDlet {

  public void startApp () {
    Display.getDisplay (this).setCurrent (new ImageCanvas ());
  }
  public void pauseApp () {}
  public void destroyApp (boolean forced) {}
}

class ImageCanvas extends Canvas {
  Image imagen;

  public ImageCanvas () {
    try {
      imagen = Image.createImage ("/icons/sun.png");
    }
    catch (IOException e) {
throw new RuntimeException ("No se puede cargar la imagen: "+e);
    }
  }

  public void paint (Graphics g) {
    g.setGrayScale (255);
    g.fillRect (0, 0, getWidth (), getHeight ());

    g.drawImage (imagen, 0, 0, Graphics.TOP | Graphics.LEFT);
    g.drawImage (imagen, getWidth () / 2, getHeight () / 2,
 Graphics.HCENTER | Graphics.VCENTER);
    g.drawImage (imagen, getWidth (), getHeight (), Graphics.BOTTOM
 | Graphics.RIGHT);
  }
}
```

Figura 121. Captura de pantalla de la emulación de la aplicación ImageTest.

3.3.4.13 Coordenadas

El sistema de coordenadas no apunta a un pixel en concreto, sino al espacio entre dos pixels, por este motivo, un rectángulo es un pixel mas alto y ancho que un rectángulo relleno. La clase Canvas nos proporciona los métodos necesarios para obtener el ancho y el alto de la pantalla a través de getWidth() y getHeight() respectivamente.

Es posible cambiar el origen de coordenadas de la pantalla mediante el método de la clase Graphics translate(int x, int y). Éste método cambia el origen de coordenadas al punto definido por los parámetros x e y provocando un desplazamiento de todos los objetos en pantalla. Este método es útil para provocar *scrolls*.

El punto 0,0 se refiere a la esquina superior izquierda, los números negativos indican hacia arriba e izquierda. Un buen ejemplo para ilustrar el uso de las coordenadas es ClipCanvas. En el cual se ha dibujado una línea recta de color gris, eligiendo en ciertos puntos puntearla y que no sea continua, con ello se puede ver claramente como se pinta sobre el lienzo del Canvas según cada coordenada.

Clase ClipCanvas.

```
package ClipCanvas;

import javax.microedition.lcdui.*;
import javax.microedition.midlet.*;

public class ClipCanvas extends MIDlet {

  public void startApp () {
    Display.getDisplay (this).setCurrent (new CanvasClip ());
  }

  public void pauseApp () {}

  public void destroyApp (boolean forced) {}
}


class CanvasClip extends Canvas {

  public void paint (Graphics g) {
    g.setGrayScale (255);
    g.fillRect (0, 0, getWidth (), getHeight ());

    int m = Math.min (getWidth (), getHeight ());
    g.setGrayScale (0);

    g.setStrokeStyle (Graphics.DOTTED);
    g.drawLine (0, 0, m, m);

    g.setClip (m / 4, m / 4, m / 2, m / 2);

    g.setStrokeStyle (Graphics.SOLID);
    g.drawLine (0, 0, m, m);
  }

}
```

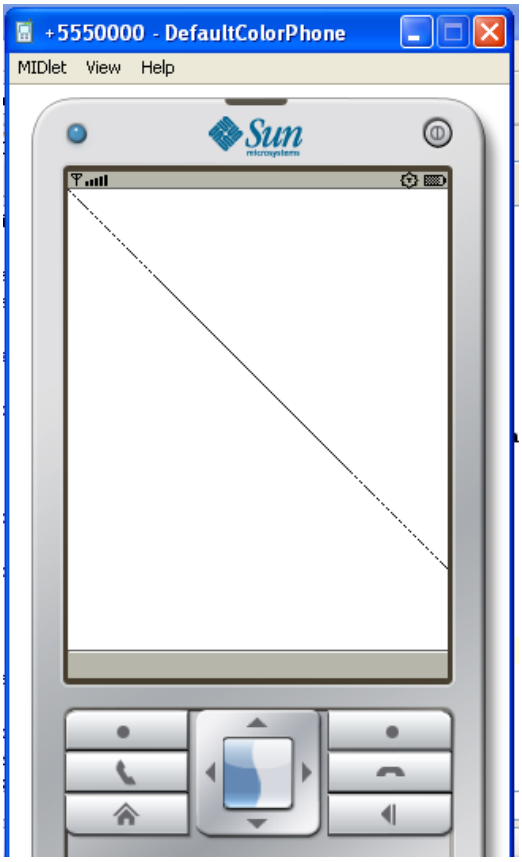


Figura 122. Captura de pantalla de la emulación de la aplicación ClipCanvas

3.3.4.14 Font

En canvas no es posible escribir, en este caso hay que ”dibujar” el texto. Los métodos más importantes son:

- drawstring()
- createFont()
- setFont()

Para ello se usa la clase Font que permitirá seleccionar el tipo de letra y almacenarlo en un objeto de este tipo para posteriormente usarlo en el Canvas.

Para seleccionar un tipo de letra hay que tener en cuenta los tres atributos que definen la fuente elegida, los cuales se pueden ver en la siguiente tabla.

	Atributos
Aspecto	FACE_SYSTEM
	FACE_MONOSPACE
	FACE_PROPORTIONAL
Estilo	STYLE_PLAIN
	STYLE_BOLD
	STYLE_ITALIC
	STYLE_UNDERLINED
Tamaño	SIZE_SMALL
	SIZE_MEDIUM
	SIZE_LARGE

Realizando combinaciones de atributos se pueden obtener bastantes tipos de fuentes donde elegir. Para crear una determinada se ha de invocar al método Font.getFont(int aspecto, int estilo, int tamaño) que nos devuelve el objeto Font deseado. Por ejemplo:

Font fuente = Font.getFont(FACE_SYSTEM,STYLE_PLAIN,SIZE_MEDIUM);

Una vez obtenido el objeto Font, se debe asociar al objeto Graphics que es el que realmente puede escribir texto en pantalla. Esto se hace de la siguiente manera:

```
g.setFont(fuente); //Seleccionamos la fuente activa.
g.drawString("Cadena de texto", getWidth()/2,
getHeight()/2,BASELINE|HCENTER);
```

Este código selecciona un tipo de fuente con la que posteriormente se escribe la cadena "Cadena de texto" posicionado en la pantalla en el punto medio (getWidth()/2, getHeight()/2). El parámetro BASELINE|HCENTER indica el posicionamiento del texto con respecto al punto medio

Para ver mas claramente el uso de esta clase se ha creado el siguiente ejemplo llamado FontTest. En este se utilizarán los métodos de la clase Font para variar el tamaño y tipo de Fuente, escribiendo la palabra Test en repetidas ocasiones. Así se puede comprobar como va variando la fuente en función de los parámetros que se le hayan pasado a los métodos. La clase FontTest como en ejemplos anteriores hereda de MIDlet y por ello para utilizar la API de bajo nivel se debe crear una subclase dentro del MIDlet que herede de Canvas.

Clase FontTest.

```
package FontTest;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


public class FontTest extends MIDlet {

  public void startApp () {
    Display.getDisplay (this).setCurrent (new FontCanvas ());
  }

  public void pauseApp () {}

  public void destroyApp (boolean forced) {}
}


class FontCanvas extends Canvas {

  static final int [] styles = {Font.STYLE_PLAIN,
                Font.STYLE_BOLD,
                Font.STYLE_ITALIC};
  static final int [] sizes = {Font.SIZE_SMALL,
                Font.SIZE_MEDIUM,
                Font.SIZE_LARGE};
  static final int [] faces = {Font.FACE_SYSTEM,
                Font.FACE_MONOSPACE,
                Font.FACE_PROPORTIONAL};

  public void paint (Graphics g) {
    Font font = null;
    int y = 0;
    g.setGrayScale (255);
    g.fillRect (0, 0, getWidth (), getHeight ());
    g.setGrayScale (0);

    for (int size = 0; size < sizes.length; size++) {
      for (int face = 0; face < faces.length; face++) {
        int x = 0;
        for (int style = 0; style < styles.length; style++) {
          font = Font.getFont
               (faces [face], styles [style], sizes [size]);
          g.setFont (font);
          g.drawString
            ("Test", x+1, y+1, Graphics.TOP | Graphics.LEFT);
          g.drawRect
            (x, y, font.stringWidth ("Test")+1,
             font.getHeight () + 1);
          x += font.stringWidth ("Test")+1;
        }
        y += font.getHeight () + 1;
      }
    }
  }
}
```

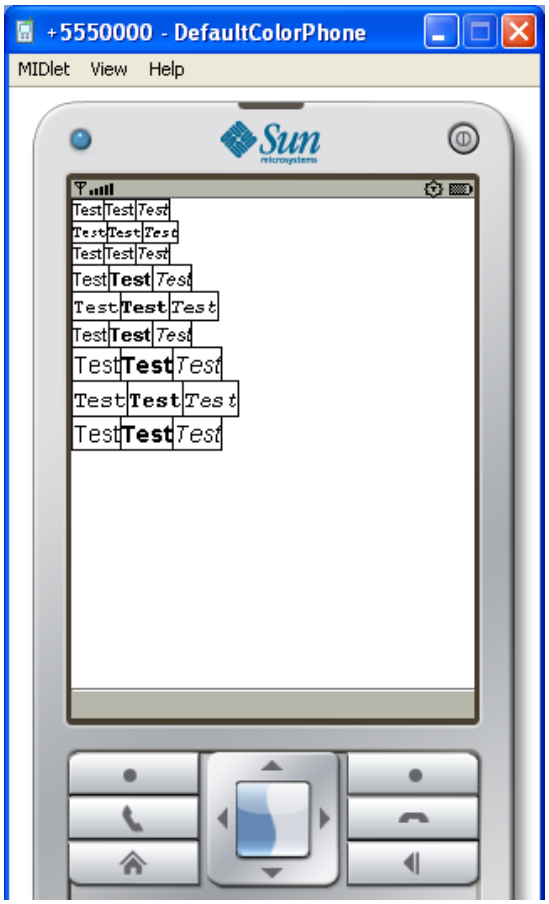


Figura 123. Captura de pantalla de la emulación de la aplicación FontTest

3.3.4.15 Animaciones

Canvas permite realizar animaciones sincronizando la aplicación para que repinte el canvas usando serviceRepaints() o callSerially(Runnable).

Canvas.callSerially() hace que se llame al método run() y puede ser utilizada desde diferentes threads.

Para no alargar más la explicación se pasará directamente a la codificación de dos ejemplos sencillos del uso de las librerías de Canvas para generar aplicaciones. Se realizará el mismo ejemplo por dos caminos diferentes. Por una parte Animacion, se creará una pequeña animación con el uso de las técnicas típicas de programación. Por otra parte FlickerTest, se realizara la misma animación anterior pero con el uso del doblebuffer el cual hace que la pantalla no parpadee al ejecutarse la aplicación. Con las imágenes que acompañaran a los ejemplos esto no es perceptible pero a la hora de emularlo en el ordenador si se nota una diferencia entre uno y otro.

Clase Animacion.

```
package Anim;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Animacion extends MIDlet {

public void startApp () {
    Display display = Display.getDisplay (this);
    display.setCurrent (new AnimacionCanvas (display, 10));
  }
  public void pauseApp () {}
  public void destroyApp (boolean forced) { }
}

class AnimacionCanvas extends Canvas implements Runnable {
  int degree = 360;
  long startTime;
  int seconds;
  Display display;

AnimacionCanvas (Display display, int seconds) {
  this.display = display;
  this.seconds = seconds;
  startTime = System.currentTimeMillis ();
}

public void paint (Graphics g) {
  g.setGrayScale (255);
  g.fillRect (0, 0, getWidth (), getHeight ());

  if (degree > 0) {
    g.setColor (255, 0, 0);
    g.fillArc (0,0, getWidth (), getHeight (), 90, degree);
    display.callSerially (this);
  }
  g.setGrayScale (0);
  g.drawArc (0, 0, getWidth ()-1, getHeight ()-1, 0, 360);
}

 public void run () {
    int permille = (int) ((System.currentTimeMillis ()  - startTime)
/ seconds);
    degree = 360 - (permille * 360) / 1000;
    repaint ();
  }
}
```

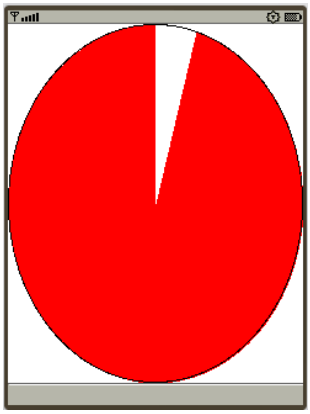


Figura 124. Captura 1 de la emulación de la aplicación Animación

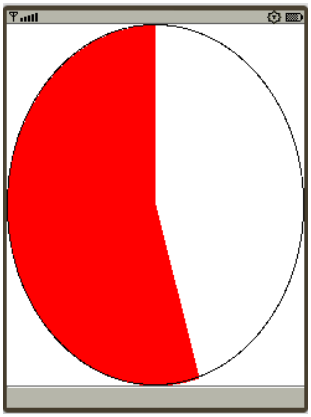


Figura 125. Captura 2 de la emulación de la aplicación Animación

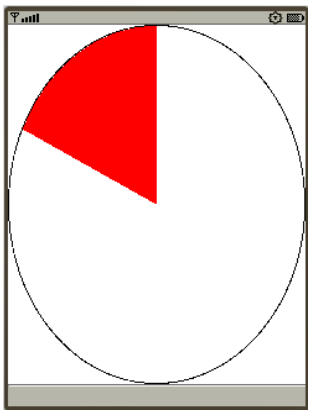


Figura 126. Captura 3 de la emulación de la aplicación Animación

Clase FlickerTest: Para esta clase no se pondrán imágenes de la emulación ya que son exactamente las mismas que para la anterior, la diferencia reside en la técnica del doblebuffer, esta es solo apreciable mediante una emulación. Se observa claramente que en este caso la pantalla no parpadea, mientras que en el anterior si.

```
package Flicker;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class FlickerTest extends MIDlet {

  public void startApp () {
    Display display = Display.getDisplay (this);
    display.setCurrent (new DobleBuffer (display, 10));
  }

  public void pauseApp () {}
  public void destroyApp (boolean forced) { }
}

class DobleBuffer extends Canvas implements Runnable {
  int degree = 360;
  long startTime;
  int seconds;
  Display display;
  Image offscreen;

  DobleBuffer (Display display, int seconds) {
    this.display = display;
    this.seconds = seconds;

 //Compuebo la pantalla del doblebuffer
    if (!isDoubleBuffered () && false)
       //Creo la imagen del doblebuffer si no ha sido creada
      offscreen = Image.createImage (getWidth (), getHeight ());
    startTime = System.currentTimeMillis ();
  }

  public void paint (Graphics g) {

    //Compruebo si puedo usar doblebuffer
    Graphics g2 = offscreen == null ? g : offscreen.getGraphics ();

    g2.setGrayScale (255);
    g2.fillRect (0, 0, getWidth (), getHeight ());

    if (degree > 0) {
      g2.setColor (255, 0, 0);
      g2.fillArc (0,0, getWidth (), getHeight (), 90, degree);
      display.callSerially (this);
    }

    g2.setGrayScale (0);
    g2.drawArc (0, 0, getWidth ()-1, getHeight ()-1, 0, 360);

 //Muestro el buffer
    if (offscreen != null)
      g.drawImage (offscreen, 0, 0, Graphics.TOP | Graphics.RIGHT);
  }
```

```
  public void run () {
    int permille = (int) ((System.currentTimeMillis () - startTime)
/ seconds);
    degree = 360 - (permille * 360) / 1000;
    repaint ();
  }
}
```

3.3.4.16 Eventos

Los eventos dentro de la clase Canvas podemos manejarlos principalmente de dos formas distintas:

• A través de Commands. Este sistema es el que hemos utilizado hasta el momento, por lo que su funcionamiento es sobradamente conocido.

• A través de códigos de teclas. Este método es el que Canvas proporciona para detectar eventos de bajo nivel. Estos códigos son valores numéricos que están asociados a las diferentes teclas de un MID. Estos códigos se corresponden con las teclas de un teclado convencional de un teléfono móvil (0-9,*,#). La clase Canvas proporciona estos códigos a través de constantes que tienen asociados valores enteros. En la siguiente tabla se muestran estos valores:

Nombre	Valor
KEY_NUM0	48
KEY_NUM1	49
KEY_NUM2	50
KEY_NUM3	51
KEY_NUM4	52
KEY_NUM5	53
KEY_NUM6	54
KEY_NUM7	55
KEY_NUM8	56
KEY_NUM9	57
KEY_STAR	42
KEY_POUND	35

Con estos códigos anteriores ya se puede conocer cual es la tecla que ha pulsado el usuario. Canvas, además proporciona unos métodos que permitirán manejar estos eventos con facilidad. La implementación de estos métodos es vacía, por lo que es misión del programador implementar los que se necesiten en la aplicación de acuerdo con el propósito de ésta. Los métodos para el manejo de códigos de teclas aparecen en la siguiente tabla.

Métodos	Descripción
boolean hasRepeatEvents()	Indica si el MID es capaz de detectar la repetición de teclas
String getKeyName(int codigo)	Devuelve una cadena de texto con el nombre del código de tecla asociado
void keyPressed(int codigo)	Se invoca cuando pulsamos una tecla
void keyReleased(int codigo)	Se invoca cuando soltamos una tecla
void keyRepeated(int codigo)	Se invoca cuando se deja pulsada una tecla

Ahora se incluirá un ejemplo para conocer exactamente como se atrapan los eventos generados por las diferentes teclas.

Este ejemplo consiste simplemente en una pantalla que muestra la tecla pulsada. Esto se consigue utilizando los eventos generados por las teclas al ser pulsadas. Exactamente como se ha visto en este apartado. Esta clase se llama KeyTest, junto con su código se incluye unas figuras con las pantallas del emulador. Como en clases anteriores se ha incluido una clase interna a KeyTest ya que esta hereda de MIDlet y hacia falta una clase que heredará de Canvas para poder manejar los eventos generados por las teclas en un Canvas.

Clase KeyTest.

```
package Key;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class KeyTest extends MIDlet {

  public void startApp () {
    Display.getDisplay (this).setCurrent (new KeyCanvas ());
  }

  public void pauseApp () {}
  public void destroyApp (boolean forced) {}
}

class KeyCanvas extends Canvas {

  String eventType = "- Presiona una tecla!";
```

```
int keyCode;

public void keyPressed (int keyCode) {
  eventType = "presionada";
  this.keyCode = keyCode;
  repaint ();
}

public void keyReleased (int keyCode) {
  eventType = "liberada";
  this.keyCode = keyCode;
  repaint ();
}

public void keyRepeated (int keyCode) {
  eventType = "repetida";
  this.keyCode = keyCode;
  repaint ();
}

public int write (Graphics g, int y, String s) {
g.drawString (s, 0, y, Graphics.LEFT|Graphics.TOP);
return y + g.getFont ().getHeight ();
}

public void paint (Graphics g) {
  g.setGrayScale (255);
  g.fillRect (0, 0, getWidth (), getHeight ());
  g.setGrayScale (0);

  int y = 0;
  y = write (g, y, "Tecla "+ eventType);
  if (keyCode == 0) return;

  y = write (g, y, "Caracter/Codigo: "+ ((keyCode < 0) ? "N/A" :
  "" +(char) keyCode) + "/" + keyCode);
  y = write (g, y, "Nombre: "+getKeyName (keyCode));

  String gameAction;

  switch (getGameAction (keyCode)) {

  case LEFT: gameAction = "IZQUIERDA"; break;
  case RIGHT: gameAction = "DERECHA"; break;
  case UP: gameAction = "ARRIBA"; break;
  case DOWN: gameAction = "ABAJO"; break;
  case FIRE: gameAction = "DISPARA"; break;
  case GAME_A: gameAction = "JUEGO_A"; break;
  case GAME_B: gameAction = "JUEGO_B"; break;
  case GAME_C: gameAction = "JUEGO_C"; break;
  case GAME_D: gameAction = "JUEGO_D"; break;
  default: gameAction = "N/A";
  }
  write (g, y, "Accion: "+gameAction);
}}
```

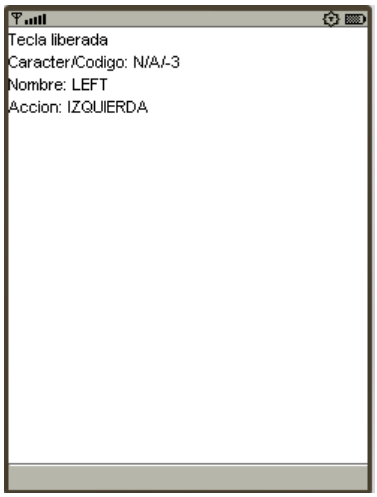


Figura 127. Captura 1 de pantalla de la emulación de la aplicación KeyTest

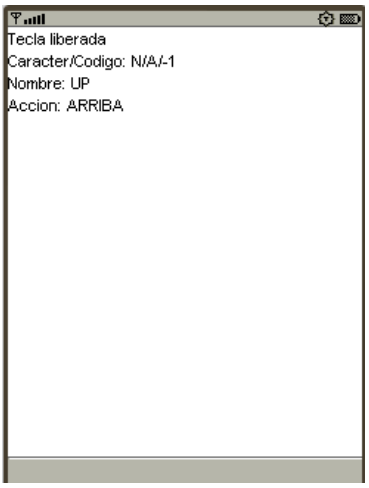


Figura 128. Captura 2 de pantalla de la emulación de la aplicación KeyTest

Las letras definidas son los cursores y el enter, al pulsar cualquiera de ellas el evento asociado a la misma salta y es posible recogerlo, analizarlo y generar una reacción para el mismo. No se ha creído necesario poner más imágenes con otras teclas pulsadas ya que con estas dos queda el funcionamiento del programa totalmente claro.

El perfil MIDP permite detectar eventos producidos por dispositivos equipados con algún tipo de puntero como un ratón o una pantalla táctil. Para ello, proporciona un conjunto de métodos cuya implementación es vacía. Estos métodos en particular, detectan las tres acciones básicas que aparecen en la siguiente tabla.

Métodos	Descripción
void pointerDragged()	Invocado cuando arrastramos el puntero
void pointerPressed()	Invocado cuando hacemos un 'click'
void pointerReleased()	Invocado cuando dejamos de pulsar el puntero

Al igual que se hacía con los eventos de teclado, se debe implementar estos métodos si queremos que el *MIDlet* detecte este tipo de eventos. Para saber si el dispositivo MID está equipado con algún tipo de puntero se hacer uso de los métodos de la siguiente Tabla.

Método	Descripción
boolean hasPointerEvents()	Devuelve *true* si el dispositivo posee algún puntero
boolean has PointerMotionEvents()	Devuelve *true* si el dispositivo puede detectar acciones como pulsar, arrastrar y soltar el puntero.

A continuación se incluirá el último de los ejemplos de este capitulo, el cual es un MIDlet capaz de detectar los eventos generados en una pantalla táctil. Para realizarlo se ha hecho uso de los métodos anteriormente mencionados. El MIDlet recibe en este caso eventos de la pantalla táctil y muestra en pantalla las coordenadas del punto donde se presiono la misma. En este caso no se puede mostrar imágenes del emulador, ya que este MIDlet necesita ser probado directamente sobre un dispositivo, porque el emulador no es capaz de percibir eventos de ratón sobre el mismo. El emulador solo funciona con las teclas, de todos modos a sido probada en el Sony Ericsson P800 y funciona perfectamente.

Para mas información acerca de la API de bajo nivel se puede consultar en la página oficial de Sun. Allí se pueden encontrar todos los métodos que no han sido incluidos en estos capítulos por no considerarlos suficientemente importantes o de una gran repercusión en el desarrollo del mismo.

Es cierto que no se han incluido paquetes como javax.microedition.lcdui.Game. El cual proporciona todos los métodos para la creación de juegos en J2ME que es una parte importante del mismo. Pero hay que decir que con lo incluido el estudio de dicho paquete y sus métodos se hace muy sencillo, ya que funciona de manera similar a lo visto aquí anteriormente.

A continuación se pondrá el código de la clase PointerTest.

Clase PointerTest

```
package Pointer;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class PointerTest extends MIDlet {

  public void startApp () {
    Display.getDisplay (this).setCurrent (new PointerCanvas ());
  }

  public void pauseApp () {}

  public void destroyApp (boolean forced) {}
}

class PointerCanvas extends Canvas {

  String eventType = "Puntero presionado!";
  int x;
  int y;

  public void pointerPressed (int x, int y) {
    eventType = "Puntero Presionado";
    this.x = x;
    this.y = y;
    repaint ();
  }

  public void pointerReleased (int x, int y) {
    eventType = "Puntero Presionado";
    this.x = x;
    this.y = y;
    repaint ();
  }

  public void pointerDragged (int x, int y) {
    eventType = "Puntero Repetido";
    this.x = x;
    this.y = y;
    repaint ();
  }

  public void paint (Graphics g) {
    g.setGrayScale (255);
    g.fillRect (0, 0, getWidth (), getHeight ());
    g.setGrayScale (0);
    g.drawString (eventType + " " +x +"/"+y,
           0, 0, Graphics.TOP|Graphics.LEFT);
    g.drawLine (x-4, y, x+4, y);
    g.drawLine (x, y-4, x, y+4);
  }
}
```

3.3.5 J2ME y Redes

3.3.5.1 Introducción

Hasta el momento se han realizado multitud de aplicaciones utilizando todas las herramientas que puede proporcionar J2ME pero siempre se han usado métodos y APIs provenientes de J2SE. Dicho de otra manera, todas las aplicaciones anteriormente realizadas podrían haberse hecho con una mayor calidad y eficiencia para J2SE. Sin embargo, aun queda un punto por explotar en J2ME, el cual es su mayor virtud, la posibilidad de los dispositivos MID de estas siempre conectados.

La posibilidad de llevar un dispositivo que ocupa poco espacio y permita realizar una comunicación en cualquier momento y lugar abre un abanico de posibles aplicaciones que no se podrían disfrutar en un PC de sobremesa, ni tan siquiera en un ordenador portátil. Piénsese en la realización de aplicaciones de mensajería instantánea o en la posibilidad de leer e-mails de una cuenta de correo o incluso enviarlos.

Este capítulo, sin duda alguna, es el más importante de todos los que se han visto hasta ahora ya que, el gran potencial de los dispositivos MID es la posibilidad de conexión en cualquier momento y transferir cualquier tipo de información mientras dure esa conexión. En este apartado se verá cómo realizar esa transferencia de información convun dispositivo MID y las herramientas que la plataforma J2ME proporciona para ese propósito. Para ello, a la vez que se va explicando cada una de estas herramientas, se va a ir creando una aplicación llamada Hogar.

Esta aplicación va a utilizar todas las herramientas vistas anteriormente y además utilizará las nuevas herramientas que se introducirán en este capitulo. La aplicación Hogar consiste en un cliente-servidor encargado de manejar una futura casa digital. Esta pensada como aplicación activa dentro de la domótica. Con ella se podrán controlar todos los aspectos de una casa digital, se podrá subir la temperatura de cualquier habitación, encender o apagar las luces, conectar el microondas y un largo etc. Pero lo mejor de la misma es que todo esto podrá realizarse desde cualquier lugar del mundo, en cualquier momento, por las grandes posibilidades que ofrece la plataforma J2ME

En esta aplicación, se va a establecer una comunicación con un *servlet* que será el encargado de responder a las peticiones y consultas y que se encargará "virtualmente" de aceptar las órdenes que le sean dadas y llevarlas a cabo. Por último, solo añadir, que en esta última aplicación deberá usarse MIDP 2.0 por las restricciones existentes en los perfiles anteriores.

3.3.5.2 Diferencias y semejanzas entre J2ME y J2SE

En ambas plataformas se usan *streams* para escribir y leer datos. Estos *streams* fueron usados en capítulos anteriores a la hora de almacenar o recuperar información en un *Record Store*. Estas clases están en el paquete java.io por lo que no forman parte del perfil MIDP. En cambio, para J2ME se dispone del paquete javax.microedition.io, el cual contiene las clases que dan soporte para el trabajo en red y comunicaciones en las aplicaciones MIDP. Este paquete existe en contraposición con el paquete java.net de J2SE.

Las aplicaciones MIDP usan los paquetes javax.microedition.io y java.io de la siguiente manera. El primer paquete contiene numerosas clases que permiten crear y manejar diferentes conexiones de red. Estas conexiones podrán usar diferentes formas de comunicación: HTTP, datagramas, sockets,… Pues bien, el paquete java.io se encargará de proporcionarnos las clases necesarias para leer y escribir en estas conexiones.

Como ya se vio en un capítulo anterior, estas clases orientadas a la conexión en red y comunicaciones reciben el nombre de *Generic Connection Framework*. En la siguiente figura se puede ver cómo se organizan jerárquicamente. Es más, también vamos a ver cómo se organizan las interfaces que suministra la especificación MIDP y que están en un nivel superior al GCF ya que implementan los detalles de diversos protocolos de comunicación.

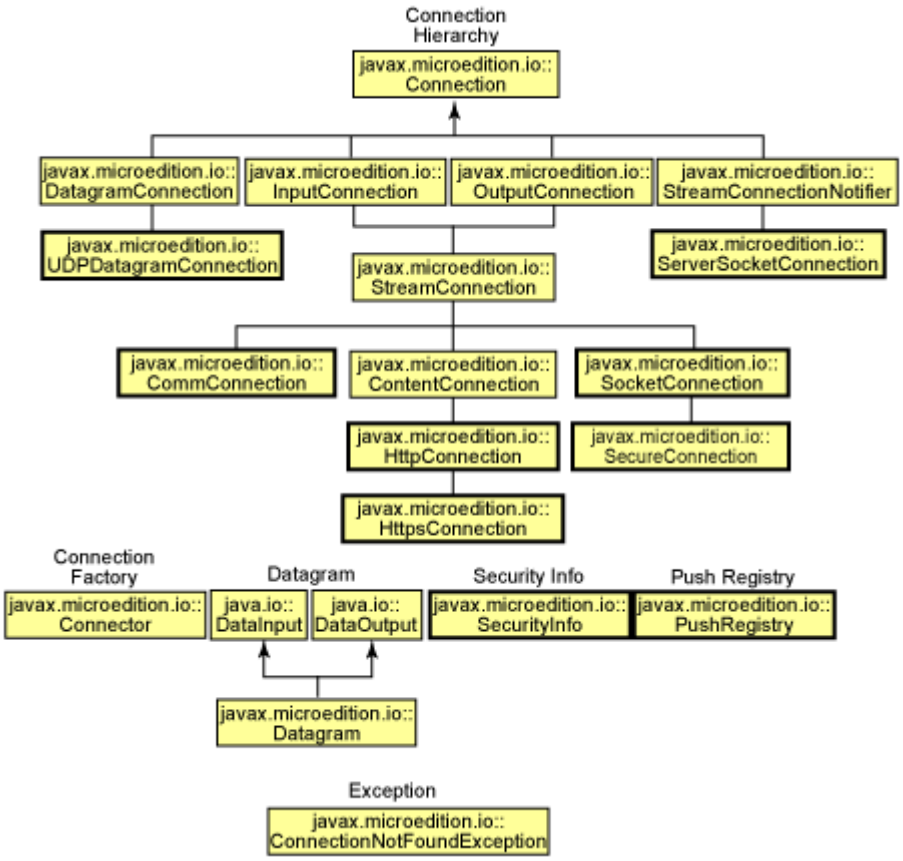


Figura 129. MIDP 2.0

La raíz del árbol es la interfaz Connection que representa la conexión más genérica y abstracta que se puede crear. El resto de interfaces que derivan de Connection representan los distintos tipos de conexiones posibles en J2ME. En lo siguientes apartados se irán viendo cada una de estas conexiones, como crearlas y manejarlas.

Mirando el anterior árbol de conexiones es fácil hacerse una idea de la filosofía que sigue el Generic Connection Framework. Al existir multitud de dispositivos MID es imposible poseer la implementación de los distintos protocolos existentes, por ello, J2ME proporciona una única clase Connector que esconde los detalles de la conexión. Esta clase puede por si misma crear cualquier tipo de conexión: Http, Socket, Datagrama, etc.

3.3.5.3 Clase Connector

public class Connector

El GCF proporciona la clase Connector que esconde los detalles de la conexión. De esta forma se puede realizar cualquier tipo de conexión usando sólo esta clase y sin preocuparse de cómo se implementa el protocolo requerido. La conexión se realiza mediante el siguiente mensaje genérico:

Connector.open("protocolo:dirección;parámetros");

Algunos ejemplos de invocación son:

Connector.open("http://direccion.es");
Connector.open("file://loquesea.exe");
Connector.open("socket://direccion:0000");

La clase Connector se encarga de buscar la clase específica que implemente el protocolo requerido. Si esta clase se encuentra, el método open() devuelve un objeto que implementa la interfaz Connection. La clase Connector posee los métodos de la siguiente tabla.

Método	Descripción
public static Connection open(String dir)	Crea y abre una conexión.
public static Connection open(String dir, int modo)	Crea y abre una conexión con permisos.
public static Connection open(String dir, int mode, boolean tespera)	Crea y abre una conexión especificando el permiso y tiempo de espera.
public static DataInputStream openDataInputStream(String dir)	Crea y abre una conexión de entrada devolviendo para ello un DataInputStream.
public static DataOutputStream openDataOutputStream(String dir)	Crea y abre una conexión de salida a través de un DataOutputStream.
public static InputStream openInputStream(String dir)	Crea y abre una conexión de entrada usando un InputStream.
public static OutputStream openOutputStream(String dir)	Crea y abre una conexión de salida devolviendo para ello un OutputStream.

En la siguiente tabla se incluyen los permisos para realizar una conexión:

Modo	Descripción
READ	Permiso de sólo lectura.
READ_WRITE	Permiso tanto de lectura como de escritura
WRITE	Permiso de sólo escritura.

Interfaz Connection

public abstract interface Connection

La interfaz Connection se encuentra en lo más alto de la jerarquía de interfaces del *Generic Connection Framework*, por lo que cualquier otra interfaz deriva de él. Una conexión de tipo Connection se crea después de que un objeto Connector invoque al método open(). Esta interfaz representa a la conexión más abstracta y genérica posible. Por esta razón, el único método que posee esta interfaz es el siguiente:

- *public void close():* Se encarga de cerrar la conexión.

Conforme se avance en la jerarquía de clases del *Generic Connection Framework* se verá que cada una de ellas va añadiendo más capacidades a la conexión.

Interfaz InputConnection

public abstract interface InputConnection extends Connection

La interfaz InputConnection representa una conexión basada en *streams* de entrada. Esta interfaz sólo posee dos métodos que devuelven objetos de tipo InputStreams:

- *DataInputStream openDataInputStream():* Devuelve un DataInputStream asociado a la conexión.
- *InputStream openInputStream():* Devuelve un InputStream asociado a la conexión

El siguiente ejemplo ilustra cómo se realiza una conexión a través de esta interfaz:

```
String url = “www.loquesea.com”;
InputConnection conexión = (InputConnection)Connector.open(url);
DataInputStream dis = conexion.openDataInputStream();
```

Interfaz OutputConnection

public abstract interface OutputConnection extends Connection

La interfaz OutputConnection representa una conexión basada en *streams* de salida. Esta interfaz sólo posee dos métodos que devuelven objetos de tipo OutputStreams:

- *DataOutputStream openDataOutputStream():* Devuelve un DataOutputStream asociado a la conexión.
- *OutputStream openOutputStream():* Devuelve un OutputStream asociado a la conexión

La conexión a través de esta interfaz se realiza de forma análoga a la interfaz InputConnection.

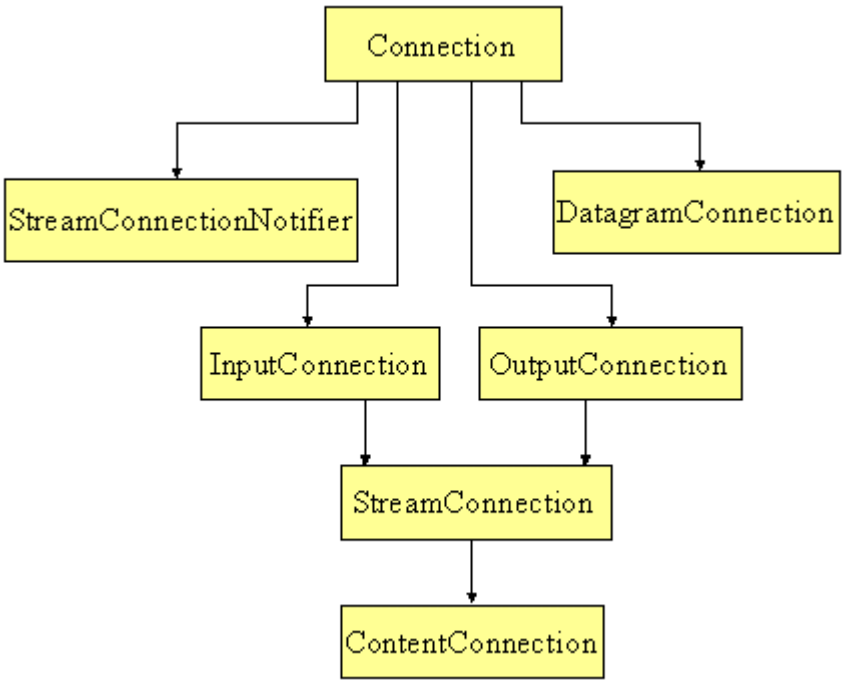


Figura 130. Diagrama con los tipos de conexiones que heredan de Connection.

Interfaz StreamConnection

public abstract interface StreamConnection extends InputConnection, OutputConnection

Esta interfaz representa una conexión basada en *streams* tanto de entrada como de salida. No añade ningún método nuevo, si no que hereda los métodos de los interfaces que están por encima de ella. Su única misión en la jerarquía del GCF es representar un tipo de conexión cuyos datos pueden ser tratados como *streams* de *bytes* y en la que es posible leer y escribir.

El siguiente ejemplo ilustra cómo se crea una conexión de este tipo en la que se va a crear un *stream* de entrada del que se leerá la información:

```
StreamConnection sc = (StreamConnection)Connector.open(url);
InputStream is = sc.openInputStream();
ByteArrayOutputStream baos = new ByteArrayOutputStream();
int c;
while((c = is.read()) != -1){
baos.write(c);
}
```

Hay que tener en cuenta que la dirección url que se usa al crear la conexión puede apuntar a un fichero (*url = "http://www.direccion.com/fichero1.txt"*) o a una imagen (*url = "http://www.direccion.com/imagen.png"*); en cualquier caso, esa información se guardará en un ByteArrayOutputStream que se puede tratar de la manera que más convenga.

Interfaz ContentConnection

public abstract interface ContentConnection extends StreamConnection

La interfaz ContentConnection extiende a la interfaz StreamConnection. En las conexiones anteriores se transmitían *bytes* sin importar su composición, pero en estas conexiones la estructura de *bytes* a transmitir debe ser conocida de antemano. Concretamente, esta interfaz representa a familias de protocolos en los que se definen atributos los cuales describen los datos que se transportan. Esta interfaz añade varios métodos que pueden ser usados en esta familia de protocolos.

- *public String getEncoding():* Devuelve la codificación empleada para representar el contenido de la información.
- *public long getLength():* Devuelve la longitud de datos.
- *public String getType():* Devuelve el tipo de datos.

Con esta interfaz se puede conocer de antemano la longitud de datos que se reciben, con lo que las operaciones de lectura de datos se pueden simplificar mucho haciendo uso de esta información. Este mecanismo de lectura se puede apreciar en el siguiente ejemplo donde se va a sustituir el bucle usado para realizar la lectura de datos por dos simples líneas de código:

```
ContentConnection cc = (ContentConnection)Connector.open(url);
is = cc.openInputStream();
byte[] datos;
int long = (int)cc.getLength();
if (long != -1) {
datos = new byte[long];
is.read(datos);
}
else // Realizar bucle de lectura de datos
```

Este algoritmo mejora al bucle del apartado anterior ya que realiza la lectura de datos de una sola vez, en vez de byte a byte.

Interfaz StreamConnectionNotifier

public abstract interface StreamConnectionNotifier extends Connection

Esta interfaz deriva directamente de Connection. Representa al establecimiento de conexiones lanzadas por clientes remotos. Si la conexión se realiza con éxito, se devuelve un StreamConnection para establecer la comunicación. Solo posee un método que es el siguiente:

- *public StreamConnection acceptAndOpen():* Devuelve un StreamConnection que representa un socket por parte del servidor.

Interfaz DatagramConnection

public abstract interface DatagramConnection extends Connection

Esta interfaz define las capacidades que debe tener una conexión basada en datagramas. A partir de esta interfaz se pueden definir distintos protocolos basados en datagramas, pero su implementación habría que realizarla a nivel del perfil. Los métodos que posee esta interfaz son estos:

Método	Descripción
public int getMaximumLength()	Devuelve la longitud máxima que puede tener un datagrama.
public int getNominalLength()	Devuelve la longitud nominal que puede tener un datagrama
public Datagram newDatagram(byte[] buf, int tam)	Crea un datagrama.
public Datagram newDatagram(byte[] buf, int tam, String dir)	Crea un datagrama.
public Datagram newDatagram(int tam)	Crea un datagrama automáticamente.
public Datagram newDatagram(int tam, String dir)	Crea un datagrama
public void receive(Datagram dat)	Recibe un datagrama.
public void send(Datagram dat)	Envía un datagrama.

3.3.5.4 Comunicaciones HTTP

El protocolo HTTP es un protocolo de tipo petición/respuesta. El funcionamiento de este protocolo es el siguiente: El cliente realiza una petición al servidor y espera a que éste le envíe una respuesta. Normalmente, esta comunicación es la que suele realizarse entre un navegador web (cliente) y un servidor web (servidor). En la aplicación que se ha realizado para este apartado llamada **Hogar**, esta comunicación se va a realizar entre un MIDlet (cliente) y un servlet (servidor) que recibirá peticiones y, dependiendo del caso, devolverá un resultado.

En la siguiente figura se pueden ver los tres estados por los que pasa una conexión http.

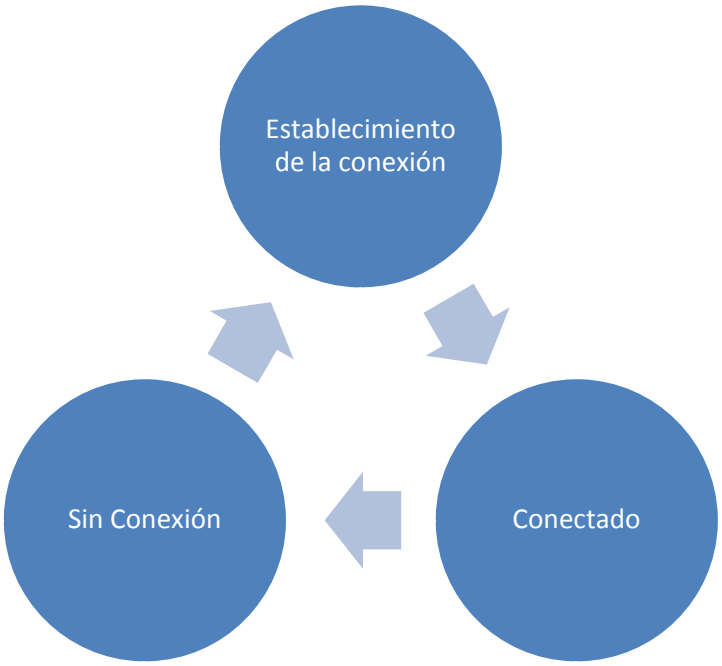


Figura 131. Estados en los que puede encontrarse una conexión http.

Estado de Establecimiento

En este estado es dónde se van a establecer los parámetros de la comunicación. El cliente prepara la petición que va a realizar al servidor, además de negociar con él una serie de parámetros como el formato, idioma, etc…

Existen dos métodos que sólo pueden ser invocados en este estado:

- *public void setRequestMethod(String tipo):* Establece el tipo de petición.
- *public void setRequestProperty(String clave, String valor):* Establece una propiedad de la petición.

El primer método establece el tipo de petición que vamos a realizar. Esta petición puede ser de los tipos indicados a continuación:

- GET: Petición de información en la que los datos se envían como parte del URL.
- POST: Petición de información en la que los datos se envían aparte en un *stream*.
- HEAD: Petición de metainformación.

El segundo método establece cierta información adicional a la petición. Esta información permite negociar entre el cliente y el servidor detalles de la petición como, por ejemplo, idioma, formato, etc. Estos campos forman parte de la cabecera de la petición y aunque existen más de 40 distintos, los más interesantes podemos verlos en la siguiente tabla.

Campo	Descripción
User-Agent	Tipo de contenido que devuelve el servidor.
Content-Language	País e idioma que usa el cliente.
Content-Length	Longitud de la petición.
Accept	Formatos que acepta el cliente.
Connection	Indica al servidor si se quiere cerrar la conexión después de la petición o se quiere dejar abierta.
Cache-Control	Sirve para controlar el almacenamiento de información.
Expires	Tiempo máximo para respuesta del servidor.
If-Modified-Since	Pregunta si el contenido solicitado se ha modificado desde una fecha dada.

Estado de Conexión

En este estado se realiza el intercambio de información entre el cliente y el servidor. En este estado es cuando el servidor envía las respuestas pertinentes al cliente.

Respuesta del servidor

Al igual que la petición del cliente poseía distintas partes, la respuesta del servidor se compone de:

- Línea de estado
- Cabecera
- Cuerpo de la respuesta.

Para conocer la respuesta del servidor, la interfaz HttpConnection proporciona diversos métodos que permitirán conocer las distintas partes de ésta. Estos son:

- *public int getResponseCode()* Devuelve el código de estado.
- *public String getResponseMessage()* Devuelve el mensaje de respuesta.

La interfaz HttpConnection dispone de 35 códigos de estado diferentes. Básicamente se pueden dividir en cinco clases de la siguiente manera:

• 1xx – Código de información.
• 2xx – Código de éxito.
• 3xx – Código de redirección.
• 4xx – Código de error del cliente.
• 5xx – Código de error del servidor.

Por ejemplo, el código 400 corresponde a la constante HTTP_BAD_REQUEST. En realidad la respuesta del servidor posee el siguiente formato:

HTTP/1.1 400 Bad Request
HTTP/1.1 200 OK

En la respuesta se incluye el protocolo usado, seguido del código de estado y de un mensaje de respuesta. Este mensaje es el devuelto al invocar el método getResponseMessage().

Al igual que el cliente puede mandar información de cabecera adicional a la petición, el servidor también puede mandar esta información al cliente. Los métodos de la interfaz HttpConnection que aparecen en la siguiente tabla se usan para conseguir esta respuesta.

Método	**Descripción**
String getHeaderField(int n)	Devuelve el valor del campo de cabecera número n.
String getHeaderField(String nombre)	Devuelve el valor del campo de cabecera especificado por el nombre.
long getHeaderFieldDate(String nombre, long def)	Devuelve un long que representa una fecha.
int getHeaderFieldInt(String nombre, int def)	Devuelve el campo nombrado como un entero.
int getHeaderFieldKey(int n)	Devuelve la clave del campo de cabecera usando un índice.
public long getDate()	Devuelve el campo de cabecera “fecha”.
public long getExpiration()	Devuelve el campo de cabecera “expires”.
public long getLastMoified()	Devuelve el campo de cabecera “last-modified”.

Existen también otros métodos que nos permiten obtener diversa información sobre la conexión:

- *String getFile()*: Devuelve el nombre del archivo de la URL.
- *String getHost():* Devuelve el host de la URL.
- *Int getPort():* Devuelve el puerto de la URL.
- *String getProtocol():* Devuelve el protocolo de la URL.
- *String getQuery():* Devuelve la cadena de petición(respuestas GET).
- *String getRef():* Devuelve la referencia.
- *String getURL():* Devuelve la cadena de conexión URL.

Estado de Cierre

La conexión entra en este estado una vez que se termina la comunicación entre el cliente y el servidor invocando al método close().

Otras Conexiones

Como hemos visto anteriormente el perfil MIDP implementa varias interfaces de comunicación además del HTTP. Ahora se van a ver a grandes rasgos estas interfaces.

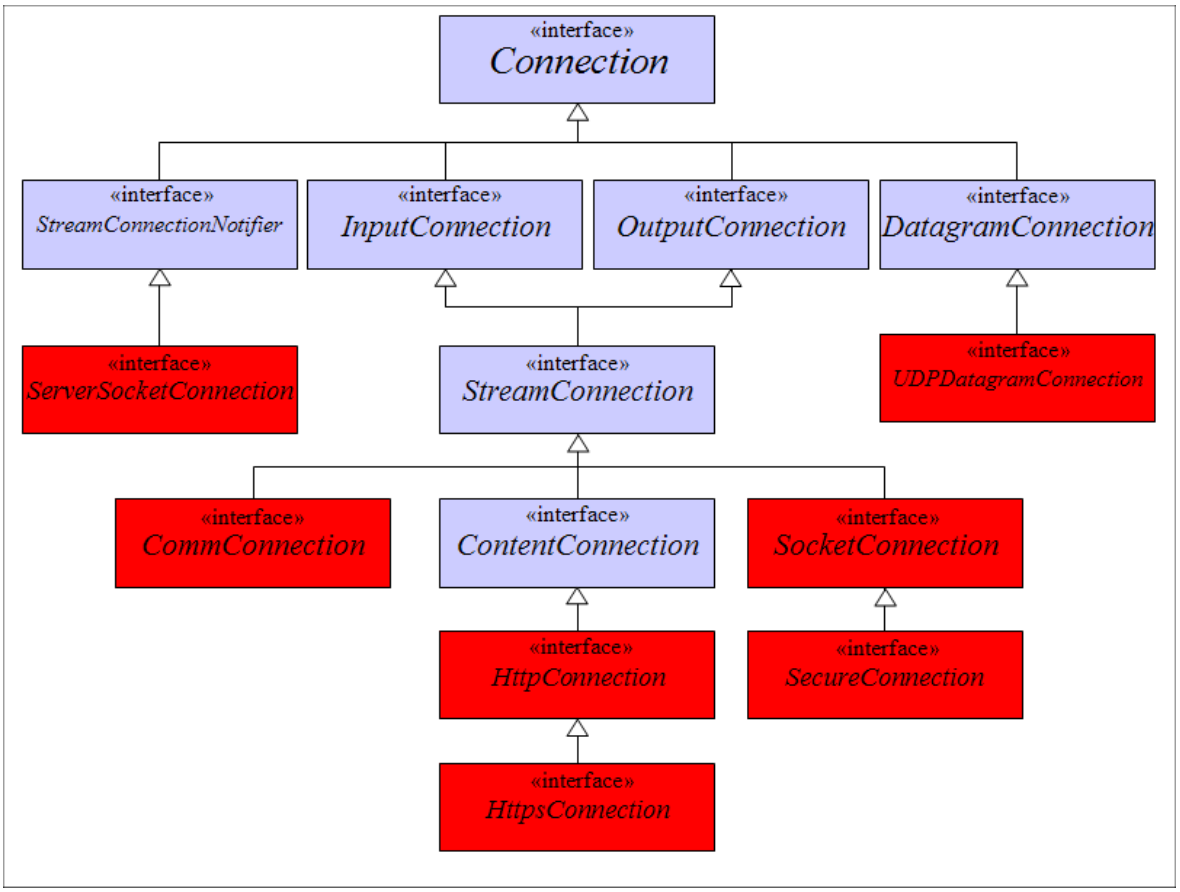


Figura 132. Diagrama de interfaces pertenecientes a la clase Connection

Interfaz HttpsConnection

public interface HttpsConnection extends HttpConnection

HTTPS es la versión segura del protocolo HTTP. Esta interfaz define los métodos necesarios para establecer una conexión de este tipo. En esta conexión, los parámetros de la petición se deben establecer antes de que ésta se envíe.

Un objeto de tipo HttpsConnection es devuelto invocando al método Connector.open(url) de la siguiente manera:

```
String url = "https://www.direccion.com";
HttpsConnection conexion = (HttpsConnection)Connector.open(url);
```

Algunos métodos de esta interfaz pueden lanzar la excepción CertificateException para indicar distintos fallos mientras se establece una conexión segura.

Interfaz UDPDatagramConnection

public interface UDPDatagramConnection extends DatagramConnection

Esta interfaz representa una conexión basada en datagramas en los que seconoce su dirección final. Esta interfaz se usa cuando el parámetro *url* del método Connector.open(url) tiene el siguiente formato:

url = "datagram://<host>:<port>";

y la conexión se realiza de la siguiente manera :

```
UDPDatagramConnection dc =
UDPDatagramConnection)Connector.open(url);
```

Si la cadena de conexión omite el *host* y el *port*, el sistema deberá de localizar un puerto libre. La dirección y el puerto local pueden conocerse usando los métodos que proporciona esta interfaz:

- *public String getLocalAddress():* Devuelve la dirección local.
- *public int getLocalPort():* Devuelve el puerto local.

Interfaz CommConnection

public interface CommConnection extends StreamConnection

Esta interfaz representa una conexión mediante un puerto serie. Tal y como su nombre indica, en esta conexión los *bits* de datos se transmiten secuencialmente, en serie. Esta conexión se establece cuando el parámetro *url* que se le pasa al método Connector.open(url) tiene el siguiente formato:

url = "comm:<port><parámetros>";

y la conexión se realiza de la siguiente manera :

CommConnection cc = (CommConnection)Connector.open(url);

Los parámetros van separados por ';' y son:

- *baudrate:* Velocidad de la conexión.
- *bitsperchar:* Número de bits por carácter.
- *stopbits:* Número de bits de parada por carácter.
- *parity:* Paridad.
- *blocking:* Estado on u off.
- *autocts:* Estado on u off
- *autorts:* Estado on u off

Por otro lado el nombre del puerto se debe indicar según el siguiente criterio:

• COM# para puertos RS-232.
• IR# para puertos IrDA IRCOMM.

donde '#' indica el número asignado al puerto.

Interfaz SocketConnection

public interface SocketConnection extends StreamConnection

Esta interfaz define una conexión entre *sockets* basados en *streams*. La conexión con el *socket* de destino tiene el siguiente formato:

url = "socket://<host>:<port>";

y la conexión se realiza de la siguiente manera :

SocketConnection sc = (SocketConnection)Connector.open(url);

Esta conexión está basada en la interfaz StreamConnection. Un StreamConnection puede ser tanto de entrada como de salida. Por otro lado si, por ejemplo, el sistema proporciona un sistema de comunicación *duplex*, para cerrar la comunicación a través del *socket* es necesario cerrar los *streams* de entrada y de salida. Si tan sólo cerramos el canal de entrada, se puede seguir mandando información por el de salida. Incluso una vez cerrados ambos, es posible volverlos a abrir si aún no se ha cerrado el *socket*.

Interfaz SecureConnection

public interface SecureConnection extends SocketConnection

Esta interfaz representa una conexión segura entre *sockets*. Esta conexión se consigue invocando al método Connector.open(url) de la siguiente forma:

String url = "ssl://<host>:<port>";

y la conexión se realiza de la siguiente manera :

SecureConnection ssc = (SecureConnection)Connector.open(url);

Si el establecimiento de la conexión falla, se lanzaría una excepción del tipo CertificateException. Esta interfaz sólo añade un método que es el siguiente:

- *public SecurityInfo getSecurityInfo():* Devuelve la información de seguridad asociada a la conexión.

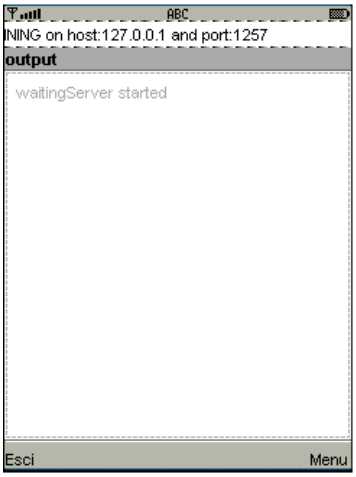


Figura 133. Captura de pantalla de una aplicación que usa la interfaz ServerSocketConnection.

Interfaz ServerSocketConnection

public interface ServerSocketConnection extends StreamConnectionNotifier

Esta interfaz representa una conexión con un servidor de *sockets*. Este tipo de conexión se establece cuando se invoca al método Connector.open(url) sin especificar el *host*:

ServerSocketConnection ssc = (ServerSocketConnection)Connector.open("socket://:00");

3.3.5.5 Desarrollo de una aplicación final a modo de ejemplo.

Ahora se pondrá el código de la aplicación desarrollada. Esta aplicación esta dividida en dos paquetes. El primero de ellos contiene los archivos fuente o clases. Estas son once clases de las que se irán poniendo sus códigos y explicando lo mas relevante de cada una. El otro paquete contiene los iconos necesarios para una perfecta visualización de la aplicación. Además de los MIDlets que se han estado utilizando hasta ahora, en esta aplicación se harán uso de SERVlets, estos son muy parecidos a los MIDlets solo que estos en lugar de correr en el cliente corren sobre un servidor. La aplicación completa recibe el nombre de HogarServlet.

Clases del cliente: En primer lugar se van a ver las clases que se ejecutaran bajo el MIDlet o cliente, todas las clases que vienen a continuación son ejecutadas desde el cliente y tienen función sobre el MIDlet puramente. Después se verán las que corren bajo el servidor y las que corren bajo el cliente pero establecen la comunicación con el servidor.

Clase Fvideo:

Esta clase es utilizada para modelar el objeto video. Su funcionalidad será similar a la de un video normal y corriente. Con esta clase se podrá grabar de un canal concreto, en el momento actual o seleccionar una hora para que empiece a grabar. Cuando sea utilizado este objeto actualizará a otra clase llamada Edificio, la cual controla todos los elementos que componen la aplicación. El funcionamiento de la clase como puede verse es muy sencillo, incluye menús para poder gestionar todas las funcionalidades del objeto, modelados a partir de formularios, listas, textfields, etc.

Código de la clase Fvideo

```
package HogarServlet;

import java.util.*;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 *
 * @author Carlos
 */
```

```
    public class Fvideo extends Form implements CommandListener,
ItemStateListener{
        private Form fecha;
        private ChoiceGroup elec;
        private TextField canal;
        private DateField fechahora;
        private Ticker tick;
        private Command atras, siguiente, aceptarfecha, aceptar;
        private boolean estado, estasig;
        private Date fechaactual;
        private Hogar midlet;

       /** Esta pantalla video nos permite seleccionar un canal para
       grabar en este momento
         o seleccionar una hora determinada para que comience a grabar*/
         public Fvideo(Hogar midlet) {

             super("Video");
             this.midlet = midlet;   //Guardo referencia del midlet
             estado = false;         //Guardo estado actual
             fechaactual = new Date();//Guardo fecha actual
             estasig = false;
             fecha = new Form("Introduzca fecha y hora");
             fechahora = new DateField("Introduce la fecha y la
            hora",DateField.DATE_TIME);
             fechahora.setDate(fechaactual);
             canal = new TextField("Seleccione el canal a
            grabar","",2,TextField.NUMERIC);
             String opc[] ={"Grabar","Programar Hora"};
             elec = new ChoiceGroup("",Choice.EXCLUSIVE,opc,null);
             tick = new Ticker("Seleccione canal y hora");

             siguiente = new Command("Siguiente",Command.OK,1);
             aceptarfecha = new Command("Aceptar",Command.OK,1);
             aceptar = new Command("Aceptar",Command.OK,1);
             atras = new Command("Atras",Command.BACK,1);
             fecha.addCommand(aceptar);
             fecha.append(fechahora);
             this.append(canal);
             this.append(elec);
             this.setTicker(tick);
             this.addCommand(atras);
             this.addCommand(aceptar);
             this.setItemStateListener(this);
             this.setCommandListener(this);
         }

       /*Si seleccionamos una hora determinada para grabar nos aparecerá
       un objeto
       *DataField en el cual podremos introducir la fecha y la hora a la
       que queremos
       *que grabe*/
         public void itemStateChanged(Item item){
             if(item == elec){
                 if(elec.getSelectedIndex() == 1){
                     this.removeCommand(aceptar);
                     this.addCommand(siguiente);
                     estasig = true;
                 }
```

```
                else{
                    if(estasig){
                        this.removeCommand(siguiente);
                        this.addCommand(aceptar);
                        estasig = false;
                    }
                }
            }
        }

          /*Cuando hayamos hecho los cambios necesarios pulsaremos
          Aceptar y con
         *el metodo siguiente (commandAction) actualizaremos la clase
         Edificio
          *que guardará el nuevo estado del video.*/
        public void commandAction(Command c, Displayable d){
            if(c==atras){
                midlet.verOpciones();
            }
            else if(c==siguiente){
                midlet.pantalla.setCurrent(fecha);
                fecha.setCommandListener(this);
                fecha.setItemStateListener(this);
            }
            else if (c==aceptar){
                midlet.edificio.setVideo(true);

midlet.edificio.setCanalVideo(Integer.parseInt(canal.getString()));
                midlet.verOpciones();
            }
        }
    }
```

Como se puede observar, esta, al igual que el resto de las clases, vienen comentadas para una mejor comprensión del código por parte del lector.

Clase Fmicroondas:

Esta clase al igual que la anterior modela un objeto puramente. Sirve para definir todos los parámetros que pueden ser utilizados por un microondas. Con esta clase se le pueden dar ordenes al objeto microondas, del cual se podrá seleccionar que se conecte a una hora determinada con una potencia y un durante un tiempo concreto. Al igual que la clase anterior una vez realizadas las peticiones que se le han hecho se actualizará mandándole la información necesaria a la clase Edificio. Esta clase hereda de Form para poder utilizar la API de alto nivel.

Código de la clase Fmicroondas:

```
package HogarServlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 *
 * @author Carlos
 */
```

```
public class Fmicroondas extends Form implements CommandListener,
ItemStateListener {
    private ChoiceGroup potencia,tiempo;
    private TextField txt;
    private Ticker tick;
    private Command atras, aceptar;
    private boolean estado;
    private int indicetxt, nivel, tiem;
    private Hogar midlet;

      /** Esta clase sirve para controlar el microondas de nuestro Hogar,
      su potencia y tiempo */
    public Fmicroondas(Hogar midlet) {
        super("Microondas");
        this.midlet = midlet;
        estado = false;
        indicetxt = -1;
        String potcad[] = {"Nivel 1", "Nivel 2", "Nivel 3", "Nivel 4"};
        String tiempocad[] = {"5 min", "10 min", "15 min", "20 min",
"Seleccionar tiempo"};

        potencia = new ChoiceGroup("Seleccione
      potencia",Choice.EXCLUSIVE,potcad,null);
        tiempo = new ChoiceGroup("Seleccione
      tiempo",Choice.EXCLUSIVE,tiempocad,null);
        txt = new TextField("Tiempo","",2,TextField.NUMERIC);
        tick = new Ticker("Seleccione potencia y tiempo");
        atras = new Command("Atras",Command.BACK,1);
        aceptar = new Command("Aceptar",Command.OK,1);
        this.append(potencia);
        this.append(tiempo);
        this.setTicker(tick);
        this.addCommand(atras);
        this.addCommand(aceptar);
        this.setCommandListener(this);
        this.setItemStateListener(this);
    }

    public void itemStateChanged(Item item){
        if (item == potencia){
            nivel = potencia.getSelectedIndex()+1;
        }
        else if(item == tiempo){
            if (this.tiempo.getSelectedIndex() == 4){
                indicetxt = this.append(txt);
                nivel = 0;
            }
            else{
                tiem = (tiempo.getSelectedIndex()+1)*5;
                if(indicetxt != -1){
                    this.delete(indicetxt);
                    indicetxt = -1;
                }
            }
        }
    }
```

```
/*Con commandAction enviamos la información de microondas actualizada a
Edificio*/
    public void commandAction(Command c, Displayable d){
        if (c == aceptar){
            midlet.edificio.setMicroondas(true);
            if(nivel != 0)
                midlet.edificio.setNivelesMic(nivel, tiem);
            else{
                String pot = txt.getString();
                midlet.edificio.setNivelesMic(nivel,
            Integer.parseInt(pot.trim()));
            }
        }
        if (indicetxt != -1){
            this.delete(indicetxt);
            indicetxt = -1;
        }
        this.potencia.setSelectedIndex(0,true);
        this.tiempo.setSelectedIndex(0,true);
        this.txt.setString("");
        midlet.verOpciones();
    }
}
```

Clase Temper:

Esta clase se encarga de controlar la temperatura de cada habitación. Con ella se puede variar la temperatura de cada estancia de la casa individualmente. Esta formada por un Gauge que permite seleccionar una temperatura dentro de un rango de valores determinado. Cuando se seleccione una temperatura para una habitación al pulsar confirmación, esta enviara la información a la clase Edificio para que actualice el registro.

Código de la clase Temper.

```
package HogarServlet;
import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Temper extends Form implements CommandListener,
ItemStateListener{
    private Command aceptar;
    private Gauge nivel;
    private Hogar midlet;
    private int temperatura;

      /** Esta clase se encargará de seleccionar la temperatura de cada
      habitación.Esta formada por un Gauge que nos permite seleccionar
      una temperatura dentro de un rango de valores.*/
    public Temper(Hogar m) {
        super("Seleccione temperatura");
        midlet = m;
        nivel = new Gauge("18 ºC",true,15,3);
        temperatura = 18;
        aceptar = new Command("Aceptar",Command.OK,1);
        this.append(nivel);
        this.addCommand(aceptar);
        this.setCommandListener(this);
        this.setItemStateListener(this);
    }
```

```
    public void itemStateChanged(Item item){
        if(item == nivel){ //Con esto controlo el Gauge
            temperatura = (nivel.getValue()+15);
            nivel.setLabel(temperatura+" ºC ");
        }
    }

    public void commandAction(Command c, Displayable d){
        if (c == aceptar){
            midlet.edificio.setTempHab(temperatura);
            midlet.setCalefeccion();
        }
    }

    public int getTemperatura(){
        return temperatura;
    }
}
```

Clase SelectAlarm:

La clase SelectAlarm da la posibilidad de activar o desactivar las alarmas. Con esta clase el usuario puede activar una alarma concreta (hay varias distribuidas por la casa ejemplo). Las alarmas en la aplicación están definidas con puntos de dos colores:

- Punto rojo: alarma desactivada.
- Punto verde: alarma activada.

Para cambiar el estado de la alarma bastará con pinchar sobre el punto en cuestión. Se ha incluido el uso de una contraseña para activar o desactivar las alarmas. Cuando se quiera activar una alarma habrá que definir una contraseña que después deberá escribirse para poder desactivarla. Esto se ha hecho utilizando un TextField y activando el campo Password del mismo. Esta clase hereda de Form para poder hacer uso de TextField.

Código de la clase SelectAlarm.

```
package HogarServlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 *
 * @author Carlos
 */
public class SelectAlarm extends Form implements CommandListener,
ItemStateListener{
    private StringItem msg;
    private ChoiceGroup elec;
    private TextField txt;
    private Ticker tick;
    private Command atras, aceptar;
    private int numtxt;
    private boolean estado;
    private int habitacionActiva = 0;
```

```
private Hogar midlet;

  /** Esta clase se encarga de activar o desactivar la alarma,
  dependiendo del estado de la misma. Nos pedirá una clave para poder
  manejarla*/
public SelectAlarm(Hogar midlet) {
    super("");
    this.midlet = midlet;  //Guardo la referencia del midlet
    numtxt = -1;  //Indice del TextField en el formulario
    tick = new Ticker("");
    txt = new TextField("Introduzca clave","",10,TextField.PASSWORD);
    String opc[]= {"No","Si"};
    elec = new ChoiceGroup("",Choice.EXCLUSIVE,opc,null);
    elec.setSelectedIndex(0,true);
    atras = new Command("Atras",Command.BACK,1);
    aceptar = new Command ("Aceptar",Command.OK,1);
    msg = new StringItem("","");
    this.setTicker(tick);
    this.append(msg);
    this.append(elec);
    this.addCommand(atras);
    this.setItemStateListener(this);
    this.setCommandListener(this);
}

public void itemStateChanged(Item item){
    if(item == elec){  //Compruebo opción elegida
        if(elec.getSelectedIndex() == 1){
            numtxt = this.append(txt);
            this.addCommand(aceptar);
        }
        else if(numtxt != -1){
            this.delete(numtxt);
            numtxt = -1;
            this.removeCommand(aceptar);
        }
    }
}

  /*Cuando pulsemos aceptar se enviara toda la información de la
  alarma a Edificio*/
public void commandAction(Command c, Displayable d){
    if(c == aceptar){
        //Compruebo contraseña
        estado = !estado;
        midlet.edificio.setAlarHab(estado);
        midlet.setAlarma();
    }
    if(numtxt != -1){
        this.delete(numtxt);
        numtxt = -1;
        this.removeCommand(aceptar);
    }
    this.txt.setString("");
    this.elec.setSelectedIndex(0,true);
    if(c == atras){
        midlet.setAlarma();
    }
}
```

```
      /*Utilizaré este método para inicializar los textos de la pantalla
      de alarma según el estado en que esta se encuentre*/
    public void setEstado(boolean est){
        estado = est;
        if(estado){
            msg.setText("¿Desea desactivar la alarma?");
            tick.setString("Alarma desactivada");
        }else{
            msg.setText("¿Desea activar la alarma?");
            tick.setString("Alarma activada");
        }
    }

    public boolean getestado(){
        return estado;
    }

    public void setHabitacion(int n){
        habitacionActiva = n;
    }
}
```

El método setEstado inicializa los textos de la pantalla de alarma dependiendo del estado en que se encuentre la misma (Activada/Desactivada).

Clase Edificio:

La clase Edificio se encarga de controlar todos los parámetros concernientes a la aplicación. Esta formada por varios arrays y matrices que guardan los estados de cada dispositivo y las coordenadas donde se encuentra cada uno. También guarda información sobre la planta del edificio y las coordenadas de sus habitaciones. Esta clase no hereda de ninguna super clase, ya que se compone de las informaciones del resto de objetos. Sirve para agrupar todos los objetos bajo el mando de la aplicación.

Codigo de la clase Edificio.

```
package HogarServlet;

import java.io.*;
import javax.microedition.lcdui.Image;
import javax.microedition.lcdui.Canvas;


/**
 *
 * @author Carlos
 */
public class Edificio {
    private Image planta;
    private int habActiva = 1;
    private int numHabitaciones;
    private int [][] adyacenciaHab;
    private int [][] coordenadasHab;
    private int [][] coordenadasTemp;
    private int [] temperatura;
```

```
private int numLuces;
private int canalvideo, nivelmicro, tiempomicro;
private boolean [] estadoLuz;
private int[][] coordenadasLuz;
private boolean [] alarma;
private boolean video = false;
private boolean microondas = false;

public Edificio() {
    numHabitaciones = getNumHabitaciones();
    try{
        planta = Image.createImage("/iconsServlet/planta.png");
        temperatura = new int[numHabitaciones];
        alarma = new boolean [numHabitaciones];
        adyacenciaHab = new int [numHabitaciones][4];
        coordenadasHab = new int [numHabitaciones][4];
        coordenadasTemp = new int [numHabitaciones][2];
    }
    catch(Exception e){
        System.out.println("No se pudo crear galeria de imagenes");
    }
}

public int[] mostrarPlanta(int mov){
    int hab = adyacenciaHab[habActiva-1][mov];
    if (hab != 0){
        habActiva = hab;
        return coordenadasHab[hab-1];
    }
    else return coordenadasHab[habActiva-1];
}

public void establecerTemperatura(int [] temp){
    for(int i=0;i<numHabitaciones;i++){
        temperatura[i] = temp[i];
    }
}

public int getTemperatura(int i){
    return temperatura[i];
}

public void setNumLuces(int n){
    numLuces = n;
    estadoLuz = new boolean[numLuces];
    coordenadasLuz = new int[numLuces][2];
}

public int getNumLuces(){
    return numLuces;
}

public void setEstadoLuz(boolean[] l){
    for(int i=0;i<numHabitaciones;i++){
        estadoLuz[i]=l[i];
    }
}

public void setCoordLuz(int [][] l){
    for(int i=0;i<numLuces;i++){
        for(int j=0;j<=1;j++){
```

```
            coordenadasLuz[i][j]= l [i][j];
        }
    }
}

public void setAlarma (boolean [] alar){
    for(int i=0;i<numHabitaciones;i++){
        alarma[i]=alar[i];
    }
}

public void setVideo(boolean est){
    video = est;
}

public void setCanalVideo(int v){//Repasar este método
    canalvideo = v;
}

public boolean getVideo(){
    return video;
}

public void setMicroondas(boolean est){
    microondas = est;
}

public boolean getMicroondas(){
    return microondas;
}

public void setNivelesMic(int n, int t){
    nivelmicro = n;
    tiempomicro = t;
}

public void establecerCoord(int[][] coord){
    for(int i=0;i<numHabitaciones;i++)
        for(int j=0;j<4;j++)
            coordenadasHab[i][j] = coord[i][j];
}

public void establecerAdyacencia(int[][] ady){
    for(int i=0;i<numHabitaciones;i++)
        for(int j=0;j<4;j++)
            adyacenciaHab[i][j]= ady[i][j];
}

public void setCoordTemp(int[][] c){
    for(int i=0;i<numHabitaciones;i++){
        for(int j=0;j<2;j++){
            coordenadasTemp[i][j]= c[i][j];
        }
    }
}

public int[][] getCoordTemp(){
    return coordenadasTemp;
}
```

```
    public void setNumHabitaciones(int habit){
        numHabitaciones = habit;
    }

    public int getNumHabitaciones(){
        return numHabitaciones;
    }

    public Image getPlanta(){
        return planta;
    }

    public int[] getCoordHabActiva(){
        return coordenadasHab[habActiva-1];
    }

    public void setTempHab(int t){
        //Enviar datos al servlet
        temperatura [habActiva-1]= t;
    }

    public boolean getAlarmaHab(){
        return alarma[habActiva-1];
    }

    public boolean[] getEstAlarma(){
        return alarma;
    }

    public void setAlarHab(boolean est){
        alarma[habActiva-1]= est;
    }
}
```

Clase Falarma:

Esta clase realmente es una pantalla que deriva de Canvas, sirve para pintar la representación de las alarmas sobre un lienzo que contendrá como fondo la planta del edificio. Como hereda de Canvas implementa el método paint(). Contiene un array con el estado de cada alarma en el método dibujarAlarmas(), una vez recorrido el array esta clase pintara las alarmas tanto en su estado activado como en su estado desactivado. Falarma también se encarga de pintar un rectángulo rojo en la habitación que esta seleccionada actualmente para que el usuario pueda saber en que punto de la pantalla esta interactuando. También incorpora el método encargado del movimiento por las distintas estancias.

Codigo de Falarma.

```
package HogarServlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 *
 * @author Carlos
 */
public class Falarma extends Canvas implements CommandListener{
    private SelectAlarm temp;
```

```
private Command aceptar,atras;
private int[][] coordTemp;
private boolean[] estAlarma;
private Image planta, conectada, noConectada;
private int[] coordRect;
private Hogar midlet;

public Falarma(Hogar midlet) {
    this.midlet = midlet;
    temp = new SelectAlarm(midlet);
    coordRect = new int[4];
    planta = midlet.edificio.getPlanta();

    //Creo los elementos del formulario principal
    atras = new Command("Atras",Command.BACK,1);
    aceptar = new Command("Aceptar",Command.OK,1);

    //Inserto los elementos el el formulario
    this.addCommand(atras);
    this.setCommandListener(this);
    try{
        conectada =Image.createImage("/iconsServlet/conec.png");
        noConectada = Image.createImage("/iconsServlet/nocon.png");
    }
    catch(Exception e){
        System.out.println("Error al crear imagenes en Falarma");
    }
}

  /*La pantalla Falarma deriva de Canvas por lo que debemos
  implementar el método paint() que será el encargado de dibujar lo
  que vemos por pantalla. En este caso dibujamos la planta del
  edificio, las alarmas y un rectángulo que será la habitación
  activa*/
public void paint(Graphics g){
    estAlarma= midlet.edificio.getEstAlarma();
    g.drawImage(planta,0,0,Graphics.TOP|Graphics.LEFT);
    dibujarAlarmas(g);
    g.setColor(255,0,0);
    g.drawRect(coordRect[0],coordRect[1],coordRect[2],coordRect[3]);
}

  /*El método dibujar alarmas recorre el array de estados y dibuja un
  circulo rojo si la alarma esta desconectada y uno verde si esta
  conectada*/
public void dibujarAlarmas(Graphics g){
    g.setColor(255,255,255);
    for(int i=0;i<midlet.edificio.getNumHabitaciones();i++){
        if(estAlarma[i]){

              g.drawImage(conectada,coordTemp[i][0],coordTemp[i][1],G
              raphics.TOP|Graphics.LEFT);
        }
        else

              g.drawImage(noConectada,coordTemp[i][0],coordTemp[i][1]
              ,Graphics.TOP|Graphics.LEFT);
    }
}

public void commandAction(Command c,Displayable d){
```

```
        if(c == atras)
            midlet.verOpciones();
    }

      /*Este método controla las acciones realizadas por el usuario. Se
      controla el movimiento del rectángulo entre las distintas
      habitaciones. Para ello utilizamos un método de la clase edificio
      que nos devuelve las nuevas coordenadas del rectángulo dependiendo
      de la tecla pulsada por el usuario.Si pulsamos el botón select
      accedemos a la pantalla de SelectAlarm.*/
    public void keyPressed(int codigo){
        int ncod = getGameAction(codigo);
        int[] ncoord;
        switch(ncod){
            case Canvas.FIRE:{
                temp.setEstado(midlet.edificio.getAlarmaHab());
                midlet.pantalla.setCurrent(temp);
                break;
            }
            case Canvas.UP:{
                ncoord = midlet.edificio.mostrarPlanta(0);
                for(int i = 0;i<4;i++)
                    coordRect[i] = ncoord[i];
                break;
            }
            case Canvas.DOWN:{
                ncoord = midlet.edificio.mostrarPlanta(1);
                for(int i=0;i<4;i++)
                    coordRect[i] = ncoord[i];
                break;
            }
            case Canvas.RIGHT:{
                ncoord = midlet.edificio.mostrarPlanta(2);
                for(int i=0;i<4;i++)
                    coordRect[i] = ncoord[i];
                break;
            }
            case Canvas.LEFT:{
                ncoord = midlet.edificio.mostrarPlanta(3);
                for(int i=0;i<4;i++)
                    coordRect[i] = ncoord[i];
                break;
            }
        }
        ncoord = null;
        repaint();
    }

    public void setCoord(int[] c){
        for(int i=0;i<midlet.edificio.getNumHabitaciones();i++)
            coordRect[i] = c[i];
    }

    public void setInfo1(int[][] c){
        coordTemp = c;
    }

    public void setInfo2(boolean[] est){
        estAlarma = est;
    }
}
```

Clase Fcalefaccion:

Fcalefaccion es otra pantalla que hereda de Canvas. En este caso se utiliza para pintar las temperaturas de cada estancia sobre la imagen planta del edificio. Su función básica es esa junto con la de actualizar las temperaturas y dar paso a la clase Temper para que las fije.

Codigo de Fcalefaccion

```
package HogarServlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 *
 * @author Carlos
 */
public class Fcalefaccion extends Canvas implements CommandListener{
    private Temper temp;
    private Gauge nivel;
    private Command atras, aceptar, acptemp;
    private int[][] coordTemp;
    private boolean estado;
    private Image planta;
    private int[] coordRect;
    private Hogar midlet;

    public Fcalefaccion(Hogar midlet) {
        this.midlet = midlet;
        temp = new Temper(midlet);
        coordRect = new int[4];
        planta = midlet.edificio.getPlanta();
        coordTemp = new int[midlet.edificio.getNumHabitaciones()][2];

        atras = new Command("Atras",Command.BACK,1);
        aceptar = new Command("Aceptar",Command.OK,1);
        this.addCommand(atras);
        this.setCommandListener(this);
    }

      /*Como este también hereda de Canvas también deberé implementar el
      método paint()*/
    public void paint(Graphics g){
        coordTemp = midlet.edificio.getCoordTemp();
        g.drawImage(planta,0,0,Graphics.TOP|Graphics.LEFT);
        dibujarTemperaturas(g);
        g.setColor(255,0,0);
        g.drawRect(coordRect[0],coordRect[1],coordRect[2],coordRect[3]);
    }

    public void dibujarTemperaturas(Graphics g){
         g.setColor(0,0,255);
        for(int i=0;i<midlet.edificio.getNumHabitaciones();i++){
            g.drawString(""+midlet.edificio.getTemperatura(i),

            coordTemp[i][0],coordTemp[i][1],Graphics.TOP|Graphics.LEFT);
        }
    }
```

```
    public void commandAction(Command c, Displayable d){
        if(c == atras){
            midlet.verOpciones();
        }
        else if(c == acptemp){
            midlet.pantalla.setCurrent(this);
            repaint();
        }
    }

    /*Esta parte de código es muy parecido al de la clase Falarma*/
     public void keyPressed(int codigo){
        int ncod = getGameAction(codigo);
        int[] ncoord;
        switch(ncod){
            case Canvas.FIRE:{
                midlet.pantalla.setCurrent(temp);
                break;
            }
            case Canvas.UP:{
                ncoord = midlet.edificio.mostrarPlanta(0);
                for(int i = 0;i<4;i++)
                    coordRect[i] = ncoord[i];
                break;
            }
            case Canvas.DOWN:{
                ncoord = midlet.edificio.mostrarPlanta(1);
                for(int i=0;i<4;i++)
                    coordRect[i] = ncoord[i];
                break;
            }
            case Canvas.RIGHT:{
                ncoord = midlet.edificio.mostrarPlanta(2);
                for(int i=0;i<4;i++)
                    coordRect[i] = ncoord[i];
                break;
            }
            case Canvas.LEFT:{
                ncoord = midlet.edificio.mostrarPlanta(3);
                for(int i=0;i<4;i++)
                    coordRect[i] = ncoord[i];
                break;
            }
        }
        ncoord = null;
        repaint();
    }

    public void setCoord(int[] c){
        for(int i=0;i<4;i++)
            coordRect[i] = c[i];
    }

    public void setCoordTemp(int[][] c){
        coordTemp = c;
    }
}
```

Clase Fluces:

Esta es otra pantalla que hereda de Canvas e implementa el método paint(). Se utiliza para pintar sobre la imagen planta del edificio las luces que componen las estancias. En esta clase simplemente se le dicen las coordenadas donde tiene que dibujar las luces. Las luces son iconos .png que pueden ser de dos tipos, apagada o encendida. Estos iconos son diferentes el uno del otro y según sea la situación de cada luz, la clase Fluces se encargará de pintarlo adecuadamente en su posición.

Codigo de la clase Fluces.

```
package HogarServlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;


/**
 *
 * @author Carlos
 */
public class Fluces extends Canvas implements CommandListener{
    private Command atras;
    private int[][] coordLuz;
    private boolean[] estLuz;
    private Image planta, encendida, apagada, selec1, selec2;
    private Hogar midlet;
    private int luzSeleccionada;

    /** Creo una nueva instancia de FLuces */
    public Fluces(Hogar midlet) {
        this.midlet = midlet;
        planta = midlet.edificio.getPlanta();
        luzSeleccionada = 0;
        atras = new Command("Atras",Command.BACK,1);
        this.addCommand(atras);
        this.setCommandListener(this);

        try{
         encendida = Image.createImage("/iconsServlet/luzencendida.png");
         apagada = Image.createImage("/iconsServlet/luzapagada.png");
         selec1 = Image.createImage("/iconsServlet/luzselec1.png");
         selec2 = Image.createImage("/iconsServlet/luzselec2.png");
        }
        catch(Exception e){
            System.out.println("Error creando imagenes en Fluces");
        }
    }

    public void paint(Graphics g){
        g.drawImage(planta,0,0,Graphics.TOP|Graphics.LEFT);
        dibujarLuces(g);
    }

    public void dibujarLuces(Graphics g){
        for(int i=0;i < midlet.edificio.getNumLuces();i++){
            if(luzSeleccionada == i){
                if(estLuz[i])
```

```
                    g.drawImage(selec2,coordLuz[i][0],coordLuz[i][1],Graphi
                    cs.TOP|Graphics.LEFT);
                else

                    g.drawImage(selec1,coordLuz[i][0],coordLuz[i][1],Graphi
                    cs.TOP|Graphics.LEFT);
            }
            else{
                if(estLuz[i])

                    g.drawImage(encendida,coordLuz[i][0],coordLuz[i][1],Gra
                    phics.TOP|Graphics.LEFT);
                else

                    g.drawImage(apagada,coordLuz[i][0],coordLuz[i][1],Graph
                    ics.TOP|Graphics.LEFT);
            }
        }
    }

    public void commandAction(Command c, Displayable d){
        if(c == atras)
            midlet.verOpciones();
    }

    public void keyPressed(int codigo){
        int ncod = getGameAction(codigo);
        int[] ncoord;
        switch(ncod){
            case Canvas.FIRE:{
                estLuz[luzSeleccionada]=!estLuz[luzSeleccionada];
                midlet.edificio.setEstadoLuz(estLuz);
                break;
            }
            case Canvas.RIGHT:{
                if(luzSeleccionada < (midlet.edificio.getNumLuces()-1))
                    luzSeleccionada++;
                break;
            }
            case Canvas.LEFT:{
                if(luzSeleccionada > 0)
                    luzSeleccionada--;
                break;
            }
        }
        repaint();
    }

    public void setInfo1(boolean[] est){
        estLuz = est;
    }

    public void setInfo2(int[][] coord){
        coordLuz = coord;
    }
}
```

Clase Hogar:

Este es el MIDlet principal en el cliente. Se encarga de utilizar todos los objetos de la aplicación y de llevar el thread principal de la misma. Hereda de MIDlet y por ello implementa sus tres métodos básicos. Además esta compuesta de multitud de métodos accesorios a los objetos que la componen. Es un ejemplo claro de polimorfismo en Java.

Codigo de la clase Hogar.

```
package HogarServlet;


import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

/**
 *
 * @author  Carlos
 * @version
 */

public class Hogar extends MIDlet implements CommandListener{
    public Display pantalla;
    public Edificio edificio;
    private Falarma alarma;
    private Fcalefaccion calefaccion;
    private Fvideo video;
    private Fmicroondas microondas;
    private Fluces luces;
    private CargaEstados cargaest;
    private Alert alerta;
    private List menu;
    private Command salir;
    private Ticker tick;

    public Hogar(){
        pantalla = Display.getDisplay(this);
        edificio = new Edificio();
        cargaest = new CargaEstados(this);
        alarma = new Falarma(this);
        calefaccion = new Fcalefaccion(this);
        video = new Fvideo(this);
        microondas = new Fmicroondas(this);

        luces = new Fluces(this);
        alerta = new Alert("¡Atención!","",null,AlertType.INFO);
        alerta.setTimeout(Alert.FOREVER);
        tick = new Ticker("Bienvenido a mi hogar");

        String opciones [] = {"Alarma", "Climatización", "Luces",
      "Video", "Microondas"};
        menu = new List("Menu",Choice.IMPLICIT,opciones,null);
        menu.setTicker(tick);
        salir = new Command("Salir",Command.EXIT,1);
        menu.addCommand(salir);
        menu.setCommandListener(this);
    }
```

```
public void startApp() {
    pantalla.setCurrent(cargaest);
}

public void pauseApp() {
}

public void destroyApp(boolean unconditional) {

    notifyDestroyed();

}

  /*Este es el menu de nuestro Midlet. Aqui seleccionaremos el
  dispositivo de nuestra casa que queramos ver o modificar.*/
public void commandAction(Command c, Displayable d){
    if(c==salir){
        destroyApp(false);
        notifyDestroyed();
    }

    else{//Si selecciono una opcion del menu
        switch(menu.getSelectedIndex()){

            case 0:{
                alarma.setCoord(edificio.getCoordHabActiva());
                pantalla.setCurrent(alarma);
                break;
            }
            case 1:{
                calefaccion.setCoord(edificio.getCoordHabActiva());
                pantalla.setCurrent(calefaccion);
                break;
            }
            case 2:{
                pantalla.setCurrent(luces);
                break;
            }
            case 3:{
                if(!edificio.getVideo())
                    pantalla.setCurrent(video);
                else{
                    alerta.setString("Video funcionando");
                    pantalla.setCurrent(alerta,menu);
                }
                break;
            }
            case 4:{
                if(!edificio.getMicroondas())
                    pantalla.setCurrent(microondas);
                else{
                    alerta.setString("Microondas    funcionando");
                    pantalla.setCurrent(alerta,menu);
                }
                break;
            }
        }
    }
}
```

```
    /**Este método lo utilizaremos para volver al menu principal desde
    otras pantallas.*/
    public void verOpciones(){
        pantalla.setCurrent(menu);
        menu.setCommandListener(this);
    }

      /*El midlet posee metodos para enviar información adicional a las
      distintas pantallas que controlan los dispositivos.Estos métodos
      nos sirven para actualizar la información de cada pantalla y que
      todo funcione bien.*/

    public void setCalefeccion(){

        pantalla.setCurrent(calefaccion);

    }

    public void setAlarma(){

        pantalla.setCurrent(alarma);

    }

    public void setInfoLuz1(boolean [] est){

        luces.setInfo1(est);

    }

    public void setInfoLuz2(int[][] coord){

        luces.setInfo2(coord);

    }

    public void setInfoAlarma1(int[][] coord){

        alarma.setInfo1(coord);

    }

    public void setInfoAlarma2(boolean[] est){

        alarma.setInfo2(est);

    }
}
```

Clases encargadas de la comunicación con el Servlet: Las dos clases que vienen a continuación CargaEstados y EnviarEstados son las encargadas de establecer la comunicación con el servlet.

Clase CargaEstados:

Esta clase se utiliza para cargar todos los estados de los dispositivos la primera vez que se ejecuta la aplicación. En esta clase están contenidos todos los parámetros de inicialización de la misma para un correcto funcionamiento. Si no se utilizaran SERVlets estos parámetros serían siempre los que se cargarían al inicio y no serían modificables. No hereda de ninguna super clase puesto que depende única y exclusivamente de ella misma. Se compone de arrays y matrices, además de un gran número de métodos de inicialización. Ha sido creada para poder recibir por parte del SERVlet el estado actual de los dispositivos y las coordenadas correspondientes, actualizándolas así en el momento preciso. Por ello esta formada por muchos métodos de petición al SERVlet, una vez recibida la información del SERVlet esta clase se encargará de actualizar la misma enviándosela a la clase Edificio.

Codigo de la clase CargaEstados.

```
package HogarServlet;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;
import java.util.*;
import java.lang.*;
import java.io.*;
import javax.microedition.io.*;

/**
 *
 * @author Carlos
 */

/*En esta clase cargo la pantalla del dispositivo, esta pantalla tiene un
gauge que ira mostrando el nivel de carga*/
public class CargaEstados extends Form implements CommandListener,
Runnable {
    private Gauge carga;
    private int indice, numluces, numHabitaciones;
    private boolean[] estluces, estAlarma;
    private int[] estCalefaccion;
    private int[][] coord, ady, coordTemp,coordLuz;
    private boolean mic, video;
    private Command cancelar, aceptar, salir;
    private Thread t;
    private Hogar midlet;

    public CargaEstados(Hogar midlet) {
        super("Cargando Información");
        this.midlet = midlet;
        carga = new Gauge("Pulse Aceptar",false,15,0);
        cancelar = new Command("Cancelar",Command.CANCEL,1);
        aceptar = new Command("Aceptar",Command.OK,1);
        salir = new Command("Salir",Command.EXIT,1);
        this.append(carga);
        this.addCommand(aceptar);
```

```
    this.setCommandListener(this);
}

public void start(){
    t = new Thread(this);
    t.start();
}

public void run(){
    carga.setLabel("Comprobando alarma");
    verEstadoAlarma();
    carga.setValue(3);
    carga.setLabel("Comprobando calefacción");
    verEstadoCalefaccion();
    carga.setValue(6);
    carga.setLabel("Comprobando luces");
    verEstadoLuz();
    carga.setValue(9);
    carga.setLabel("Comprobando microondas");
    verEstadoMicroondas();
    carga.setValue(12);
    carga.setLabel("Comprobando video");
    verEstadoVideo();
    carga.setValue(15);
    carga.setLabel("Estableciendo coordenadas");
    setCoordenadas();
    carga.setLabel("Carga Completa");
    try{
        t.sleep(1000);
    }
    catch(Exception e){
        System.out.println("Error en la carga y comprobación de
        componentes");
    }
    midlet.verOpciones();
}

public void commandAction(Command c, Displayable d){
    if(c == cancelar){
        //No hago nada
    }
    else if(c == aceptar){
        this.removeCommand(aceptar);
        this.addCommand(cancelar);
        this.start();
    }
}

  /*Los métodos siguiente sirven para comprobar los estados de cada
  dispositivo. En este caso comprobaremos el estado de la
  alarma.Llamo al método cargarAlarma() que es que realiza la
  conexión con el Servlet y le realiza la petición correcta. Al
  finalizar cargarAlarma() ya tenemos los estados del servidor
  cargados en el array estAlarma[], además del número de habitaciones
  en la variable numHabitaciones. Envio esta información a la clase
  Edificio.*/
public void verEstadoAlarma(){
    cargarAlarma();
    midlet.edificio.setNumHabitaciones(numHabitaciones);
    midlet.edificio.setAlarma(estAlarma);
    midlet.setInfoAlarma2(estAlarma);
```

```
        try{
            t.sleep(1000);
        }
        catch(Exception e){
            System.out.println("Error en estado alarma");
        }
    }

    public void verEstadoCalefaccion(){
        cargarCalefaccion();
        midlet.edificio.establecerTemperatura(estCalefaccion);
        try{
            t.sleep(1000);
        }
        catch(Exception e){
            System.out.println("Error en estado calefacción");
        }
    }

    public void verEstadoMicroondas(){
        cargarMic();
        midlet.edificio.setMicroondas(mic);
        try{
            t.sleep(1000);
        }
        catch(Exception e){
            System.out.println("Error en estado microondas");
        }
    }

    public void verEstadoVideo(){
        cargarVideo();
        midlet.edificio.setVideo(video);
         try{
            t.sleep(1000);
        }
        catch(Exception e){
            System.out.println("Error en estado video");
        }
    }

    public void verEstadoLuz(){
        cargarNumLuces();
        midlet.edificio.setNumLuces(numluces);  //Envio el número de
luces
        midlet.edificio.setEstadoLuz(estluces);
        midlet.setInfoLuz1(estluces);
    }

    public void setCoordenadas(){
        try{
            cargarCoordenadas();
            midlet.edificio.establecerAdyacencia(ady);
            midlet.edificio.establecerCoord(coord);
            midlet.edificio.setCoordTemp(coordTemp);
            midlet.edificio.setCoordLuz(coordLuz);
            midlet.setInfoLuz2(coordLuz);
            midlet.setInfoAlarma1(coordTemp);
            coord = null;
            ady = null;
            coordLuz = null;
```

```
            coordTemp = null;
        }
        catch(Exception e){
            System.out.println("Error al establecer coordenadas");
        }
    }
      /*Aqui realizo la conexión con el servidor y envio con la URL la
      petición adecuada. Se inicializan los campos de cabecera y se
      procesa la petición a traves del método procesar(). El primer
      parámetro que recibe este método indica el tipo de petición
      enviada*/
    public void cargarNumLuces(){
        try{
            String url = "http://localhost:8090/proyecto/Hogar?accion=0";
            HttpConnection hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");
            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");
            hc.setRequestProperty("Content-Type","aplication/octet-
            stream");
            hc.setRequestProperty("Connection","close");
            hc.setRequestMethod(HttpConnection.GET);
            System.out.println("Conexión establecida y petición
            enviada");
            InputStream is = hc.openInputStream();
            procesar(0,hc,is);
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

    public void cargarMic(){
        try{
            String url = "http://localhost:8090/proyecto/Hogar?accion=1";
            HttpConnection hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");
            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");
            hc.setRequestProperty("Content-Type","aplication/octet-
            stream");
            hc.setRequestProperty("Connection","close");
            hc.setRequestMethod(HttpConnection.GET);
            System.out.println("Conexión establecida y petición
            enviada");
            InputStream is = hc.openInputStream();
            procesar(1,hc,is);
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

    public void cargarVideo(){
        try{
            String url = "http://localhost:8090/proyecto/Hogar?accion=2";
            HttpConnection hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");
            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");
```

```
            hc.setRequestProperty("Content-Type","aplication/octet-
            stream");
            hc.setRequestProperty("Connection","close");
            hc.setRequestMethod(HttpConnection.GET);
            System.out.println("Conexión establecida y petición
            enviada");
            InputStream is = hc.openInputStream();
            procesar(2,hc,is);
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

    public void cargarAlarma(){
        try{
            String url = "http://localhost:8090/proyecto/Hogar?accion=3";
            HttpConnection hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");
            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");
            hc.setRequestProperty("Content-Type","aplication/octet-
            stream");
            hc.setRequestProperty("Connection","close");
            hc.setRequestMethod(HttpConnection.GET);
            System.out.println("Conexión establecida y petición
            enviada");
            InputStream is = hc.openInputStream();
            procesar(3,hc,is);
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

    public void cargarCalefaccion(){
        try{
            String url = "http://localhost:8090/proyecto/Hogar?accion=4";
            HttpConnection hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");
            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");
            hc.setRequestProperty("Content-Type","aplication/octet-
            stream");
            hc.setRequestProperty("Connection","close");
            hc.setRequestMethod(HttpConnection.GET);

            System.out.println("Conexión establecida y petición
            enviada");
            InputStream is = hc.openInputStream();
            procesar(4,hc,is);
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

    public void cargarCoordenadas(){
        try{
            String url = "http://localhost:8090/proyecto/Hogar?accion=5";
            HttpConnection hc = (HttpConnection)Connector.open(url);
```

```
            hc.setRequestProperty("Content-Language","es-ES");
            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");
            hc.setRequestProperty("Content-Type","aplication/octet-
            stream");
            hc.setRequestProperty("Connection","close");
            hc.setRequestMethod(HttpConnection.GET);
            System.out.println("Conexión establecida y petición
            enviada");
            InputStream is = hc.openInputStream();
            procesar(5,hc,is);
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

      /*Con este método realizo la lectura de la información recibida y
      la guardo en un array*/
        public void procesar(int tipoPeticion, HttpConnection http,
        InputStream inst) throws IOException{
            try{
                if(http.getResponseCode() == HttpConnection.HTTP_OK){
                    System.out.println("OK");
                    int lon =(int) http.getLength();
                    byte datos[];
                    if(lon != -1){
                        System.out.println("Longitud Conocida");
                        datos = new byte[lon];
                        System.out.println("Creado array de lon"+lon);
                        inst.read(datos,0,lon);
                        System.out.println("Datos leidos");
                    }
                    else{
                        System.out.println("Longitud no conocida");
                       ByteArrayOutputStream baos = new
                       ByteArrayOutputStream();
                        int ch;
                        while((ch = inst.read()) != -1)
                            baos.write(ch);
                        datos = baos.toByteArray();
                        baos.close();
                    }

/*En esta parte voy a tratar la información recibida, segun el tipo de
petición actualizaré las variables de los dispositivos correctamente.*/
                    ByteArrayInputStream bais = new ByteArrayInputStream
                    (datos);
                    DataInputStream dis = new DataInputStream(bais);
                    switch(tipoPeticion){
                        case 0:{
                            numluces = dis.readInt();
                            estluces = new boolean[numluces];
                            for(int i=0;i<numluces;i++){
                                estluces[i] = dis.readBoolean();
                            }
                            break;
                        }
                        case 1:{
                            mic = dis.readBoolean();
                            break;
```

```
                        }
                        case 2:{
                            video = dis.readBoolean();
                            break;
                        }
                        case 3:{
                            numHabitaciones = dis.readInt();
                            estAlarma = new boolean[numHabitaciones];
                            for(int i = 0;i<numHabitaciones;i++){
                                estAlarma[i]=dis.readBoolean();
                            }
                            break;
                        }
                        case 4:{
                            estCalefaccion = new int[numHabitaciones];
                            for(int i = 0;i<numHabitaciones;i++){
                                estCalefaccion[i]= dis.readInt();
                            }
                            break;
                        }
                        case 5:{
                            coord = new int[numHabitaciones][4];
                            for(int i=0;i<numHabitaciones;i++){
                                for(int j=0;j<4;j++){
                                    coord[i][j]= dis.readInt();
                                }
                            }
                            ady = new int[numHabitaciones][4];
                            for(int i=0;i<numHabitaciones;i++){
                                for(int j=0;j<4;j++){
                                    ady[i][j]= dis.readInt();
                                }
                            }
                            coordTemp= new int[numHabitaciones][2];
                            for(int i=0;i<numHabitaciones;i++){
                                for(int j=0;j<2;j++){
                                    coordTemp[i][j]= dis.readInt();
                                }
                            }
                            coordLuz = new int[numluces][2];
                            for(int i=0;i<numluces;i++){
                                for(int j=0;j<2;j++){
                                    coordLuz[i][j]= dis.readInt();
                                }
                            }
                            break;
                        }
                    }
                    dis.close();
                    bais.close();
                }
            }
            catch(Exception e){
                System.out.println("Error en método procesar");
            }
        }
}
```

Clase EnviarEstados: Esta clase posee métodos encargados de enviar al SERVlet los estados de los distintos dispositivos. Estos métodos tienen como nomenclatura sendNombreDispositivo. Lo único que hace cada método es conectarse al servlet usando distintos parámetros en la URL.

Código de la clase EnviarEstados.

```
package HogarServlet;

import java.io.*;
import javax.microedition.io.*;

/**
 *
 * @author Carlos
 */

public class EnviarEstados {
    private HttpConnection hc;
    private StringBuffer cad;

    /** Creates a new instance of EnviarEstados */
    public EnviarEstados() {
        hc = null;
        cad = null;
    }

    public void sendTemperatura(int hab, int temp){
        try{
            cad = new StringBuffer(
            http://localhost:8090/proyecto/Hogar?accion=6&tipo=1& );

            String chab = "p1="+hab;
            String ctemp = "&p2="+temp;
            cad.append(chab);
            cad.append(ctemp);
            String url = cad.toString();
            hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");

            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");

            hc.setRequestMethod(HttpConnection.GET);
            if(hc.getResponseCode()==HttpConnection.HTTP_OK)
                  System.out.println("Petición enviada correctamente");

            else
                  System.out.println("Petición no recibida");
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

    public void sendEstAlarma(int hab,boolean alar){
        try{
            cad = new StringBuffer(
            http://localhost:8090/proyecto/Hogar?accion=6&tipo=2& );
```

```
            String chab = "p1="+hab;
            String calar = "&p2="+alar;
            cad.append(chab);
            cad.append(calar);
            String url = cad.toString();
            hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");

            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");

            hc.setRequestMethod(HttpConnection.GET);
            if(hc.getResponseCode()==HttpConnection.HTTP_OK)

                  System.out.println("Petición enviada correctamente");
            else
                  System.out.println("Petición no recibida");
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

    public void sendEstLuz(int luz, boolean est){
         try{
            cad = new StringBuffer(
            http://localhost:8090/proyecto/Hogar?accion=6&tipo=3& );

            String cluz = "p1="+luz;
            String cest = "&p2="+est;
            cad.append(cluz);
            cad.append(cest);
            String url = cad.toString();
            hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");

            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");

            hc.setRequestMethod(HttpConnection.GET);
            if(hc.getResponseCode()==HttpConnection.HTTP_OK)
                  System.out.println("Petición enviada correctamente");
            else
                  System.out.println("Petición no recibida");
        }
        catch(Exception e){
            System.out.println(e);
        }
    }

    public void sendEstMic(int nivel,int tiempo){
        try{
            cad = new StringBuffer(
            http://localhost:8090/proyecto/Hogar?accion=6&tipo=4& );

            String cnivel = "p1="+nivel;
            String cpot = "&p2="+tiempo;
            cad.append(cnivel);
            cad.append(cpot);
            String url = cad.toString();
            hc = (HttpConnection)Connector.open(url);
```

```
            hc.setRequestProperty("Content-Language","es-ES");

            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");

            hc.setRequestMethod(HttpConnection.GET);

            if(hc.getResponseCode()==HttpConnection.HTTP_OK)

                  System.out.println("Petición enviada correctamente");
            else

                  System.out.println("Petición no recibida");
        }

            catch(Exception e){

                  System.out.println(e);
        }
    }

    public void sendEstVideo(int cadena){
        try{
            cad = new StringBuffer(
            http://localhost:8090/proyecto/Hogar?accion=6&tipo=5& );

            String ccad = "p1="+cadena;
            cad.append(ccad);
            String url = cad.toString();
            hc = (HttpConnection)Connector.open(url);
            hc.setRequestProperty("Content-Language","es-ES");

            hc.setRequestProperty("User-Agent","Profile/MIDP-2.0
            Configuration/CLDC-1.0");

            hc.setRequestMethod(HttpConnection.GET);

            if(hc.getResponseCode()==HttpConnection.HTTP_OK)

                  System.out.println("Petición enviada correctamente");
            else

                  System.out.println("Petición no recibida");
        }

        catch(Exception e){

                  System.out.println(e);
        }
    }

}
```

Clases del Servidor: A continuación se van a ver la clase que funcionan bajo un servidor, en este caso será Hogar. Ya se vio anteriormente otra clase también llamada Hogar, pero esta que se verá ahora es su versión para el servidor.

Clase Hogar del servidor:

Esta clase es similar al MIDlet que se vio anteriormente, solo que ya no es un MIDlet, ahora correrá bajo un servidor y será un SERVlet (los SERVlet son aplicaciones que funcionan sobre un servidor). Por este motivo esta clase se compondrá ahora de muchas más variables que debe manejar. En esencia es la misma clase que la anterior, pero con los añadidos para las diferentes conexiones. Como puede observarse, el *servlet* lleva codificadas las coordenadas de posicionamiento de los componentes; indudablemente, una buena práctica de programación llevaría a introducir dichos datos en un fichero de propiedades, con el fin de hacer el *servlet* más parametrizable. No obstante, el actual objetivo se centra en estudiar las técnicas de programación con J2ME, motivo por el que se ha optado por este mecanismo de programación.

Código de la clase Hogar del servidor.

```
package HogarServlet;

     import javax.servlet.http.*;
     import javax.servlet.*;
     import java.io.*;

     public class Hogar extends HttpServlet {

     private int numLuces, numHabitaciones;
     private int[] estadoCalefaccion;
     private boolean[] estadoLuces, estadoAlarma;
     private boolean estadoMic, estadoVideo;
     private int[][] coord, ady, coordTemp, coordluz;

     public void init(ServletConfig c){
      try{
            super.init(c);
            numLuces = 6;
            numHabitaciones = 4;
            estadoLuces = new boolean [numLuces];
            for (int i=0;i<numLuces;i++){
            estadoLuces[i]=true;
      }

      estadoMic = true;
      estadoVideo = true;
      estadoAlarma = new boolean[numHabitaciones];
      estadoCalefaccion = new int[numHabitaciones];

      for (int i=0;i<numHabitaciones;i++){

            estadoAlarma[i]=true;
            estadoCalefaccion[i]=19;
      }

            coord = new int[numHabitaciones][4];
            coord[0][0] = 9;coord[0][2] = 86;
```

```
      coord[0][1] = 12;coord[0][3] = 47;
      coord[1][0] = 98;coord[1][1] = 14;
      coord[1][2] = 77;coord[1][3] = 83;
      coord[2][0] = 11;coord[2][1] = 65;
      coord[2][2] = 84;coord[2][3] = 93;
      coord[3][0] = 100;coord[3][1] = 102;
      coord[3][2] = 73;coord[3][3] = 56;

      ady = new int[numHabitaciones][4];
      ady[0][0] = 0;ady[0][1] = 3;
      ady[0][2] = 2;ady[0][3] = 0;
      ady[1][0] = 0;ady[1][1] = 4;
      ady[1][2] = 0;ady[1][3] = 1;
      ady[2][0] = 1;ady[2][1] = 0;
      ady[2][2] = 4;ady[2][3] = 0;
      ady[3][0] = 2;ady[3][1] = 0;
      ady[3][2] = 0;ady[3][3] = 3;

      coordTemp = new int[numHabitaciones][2];
      coordTemp[0][0] = 45;coordTemp[0][1] = 44;
      coordTemp[1][0] = 128;coordTemp[1][1] = 80;
      coordTemp[2][0] = 47;coordTemp[2][1] = 72;
      coordTemp[3][0] = 130;coordTemp[3][1] = 106;

      coordluz = new int[numLuces][2];
      coordluz[0][0] = 45;coordluz[0][1] = 36;
      coordluz[1][0] = 103;coordluz[1][1] = 20;
      coordluz[2][0] = 132;coordluz[2][1] = 49;
      coordluz[3][0] = 64;coordluz[3][1] = 73;
      coordluz[4][0] = 55;coordluz[4][1] = 105;
      coordluz[5][0] = 150;coordluz[5][1] = 116;
  }
  catch(Exception e){
  }
}

public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{

  try{
        String parametro = request.getParameter("accion");
             If (parametro.compareTo("6") == 0){
             recibir(request);
             response.setStatus(response.SC_OK);
        }
        else send(response,parametro);
  }
  catch(Exception e){
  }
}
public void doPost(HttpServletRequest request,HttpServletResponse
response) throws ServletException, IOException {}

private void send(HttpServletResponse response, String parametro)
throws Exception{

       byte[] data;// = new byte[0];
       data = serialize(parametro);
       response.setStatus(response.SC_OK);
       response.setContentLength(data.length);
       response.setContentType("application/octect-stream");
```

```
      OutputStream os = response.getOutputStream();
      os.write(data);
      os.close();
}

private byte[] serialize(String parametro) throws IOException{

      ByteArrayOutputStream bout = new ByteArrayOutputStream();
      DataOutputStream dout = new DataOutputStream(bout);

      if (parametro.compareTo("0") == 0){

            dout.writeInt(numLuces);
            for (int i=0;i<numLuces;i++){
            dout.writeBoolean(estadoLuces[i]);
       }

 }
 if (parametro.compareTo("1") == 0){

       dout.writeBoolean(estadoMic);

 }
 if (parametro.compareTo("2") == 0){

       dout.writeBoolean(estadoVideo);

 }
 if (parametro.compareTo("3") == 0){

       dout.writeInt(numHabitaciones);
       for (int i=0;i<numHabitaciones;i++){

             dout.writeBoolean(estadoAlarma[i]);
       }
 }
 if (parametro.compareTo("4") == 0){

       for (int i=0;i<numHabitaciones;i++){

             dout.writeInt(estadoCalefaccion[i]);
       }
 }
 if (parametro.compareTo("5") == 0){

       for (int i=0;i<numHabitaciones;i++){
             for (int j=0;j<4; j++){

                   dout.writeInt(coord[i][j]);
             }
       }
       for (int i=0;i<numHabitaciones;i++){
             for (int j=0;j<4; j++){

                   dout.writeInt(ady[i][j]);
             }
       }
       for (int i=0;i<numHabitaciones;i++){
             for (int j=0;j<2; j++){

                   dout.writeInt(coordTemp[i][j]);
```

```
                }
            }
            for (int i=0;i<numLuces;i++){
                for (int j=0;j<2; j++){

                    dout.writeInt(coordluz[i][j]);
                }
            }
        }
            dout.flush();
        return bout.toByteArray();
      }

      public void recibir(HttpServletRequest req){

        String tipoPet = req.getParameter("tipo");
        if (tipoPet.compareTo("1") == 0){

            String hab = req.getParameter("p1");
            String temp = req.getParameter("p2");
            estadoCalefaccion[Integer.parseInt(hab)] = Integer.parseInt
            (temp);
        }
        if (tipoPet.compareTo("2") == 0){

            String hab = req.getParameter("p1");
            String est = req.getParameter("p2");
            estadoAlarma[Integer.parseInt(hab)] = (Boolean.valueOf
            (est)).booleanValue();
        }
        if (tipoPet.compareTo("3") == 0){

            String luz = req.getParameter("p1");
            String est = req.getParameter("p2");
            estadoLuces[Integer.parseInt(luz)] = (Boolean.valueOf
            (est)).booleanValue();
        }
        if (tipoPet.compareTo("4") == 0){

            String nivel = req.getParameter("p1");
            String tiempo = req.getParameter("p2");
            estadoMic = true;
            //Establecer potencia y tiempo del microondas
        }
        if (tipoPet.compareTo("5") == 0){

            String cadena = req.getParameter("p1");
            estadoVideo = true;

            //Establecer grabación de la cadena elegida

        }
    }
}
```

3.3.5.6 Emulación de la aplicación HogarServlet

Ahora se realizará una emulación de la aplicación que se ha creado y se irán poniendo imágenes con copias de pantalla de la misma. De esta manera se ira viendo el resultado final de la misma y explicando cada uno de los menús. Cabe resaltar que para la correcta emulación de esta aplicación se ha tenido que utilizar una herramienta que incorpora Netbeans que no se había utilizado hasta ahora. Esta herramienta es Tomcat. Tomcat pertenece al servidor Apache y se utiliza en este caso para cargar el SERVlet sobre el mismo y asi poder emular la transmisión de datos entre servidor y cliente.

Esta herramienta se puede encontrar en la parte superior izquierda, junto a la pestaña Project y Files hay una tercera pestaña llamada Runtime. En ella se pueden encontrar todas las herramientas para las diversas ejecuciones que se deseen hacer de una aplicación en Netbeans, entre ellas esta Tomcat.

Ahora se pondrán las imágenes del resultado de la emulación:

Figura 134. Menú de lanzamiento de la aplicación HogarServlet

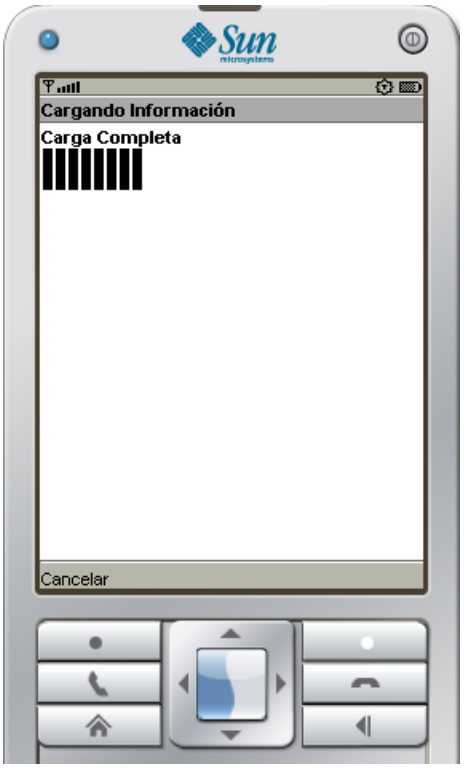


Figura 135. Menú de carga de variables (inicio aplicación)

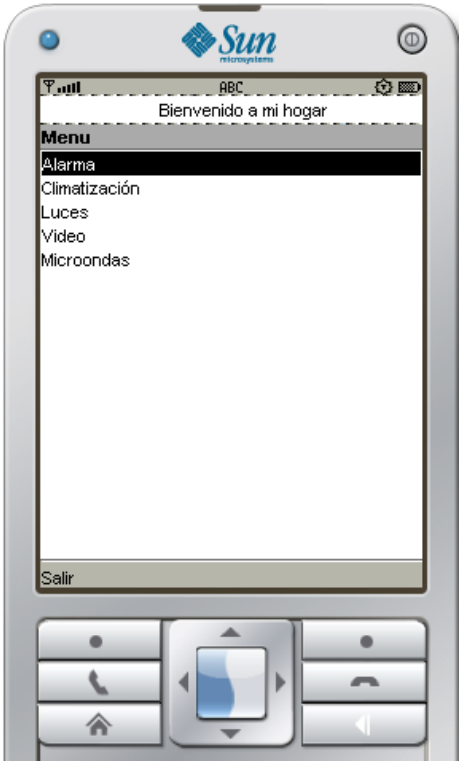


Figura 136. Menú principal de la aplicación

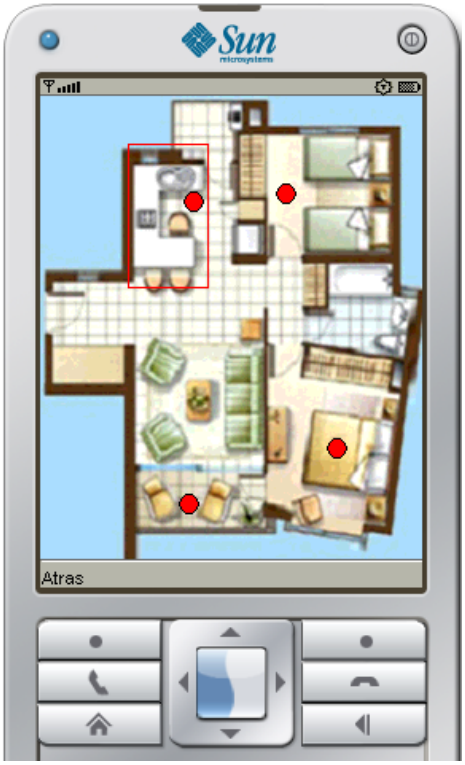


Figura 137. Submenú Alarma (muestra pantalla de alarmas)

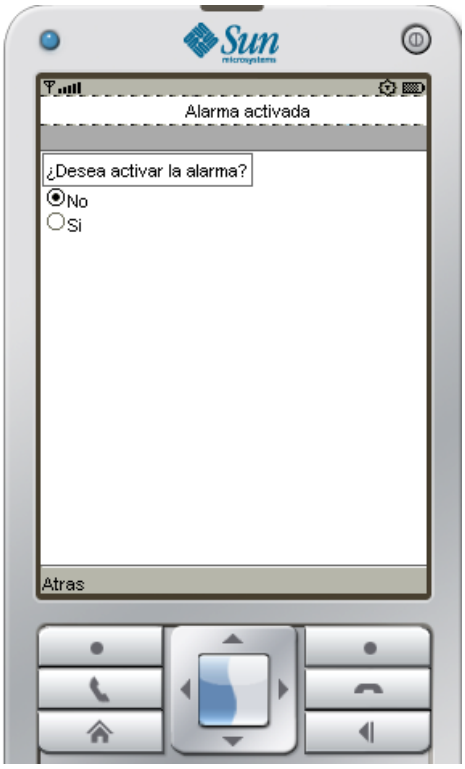


Figura138. Menú de activación de alarma

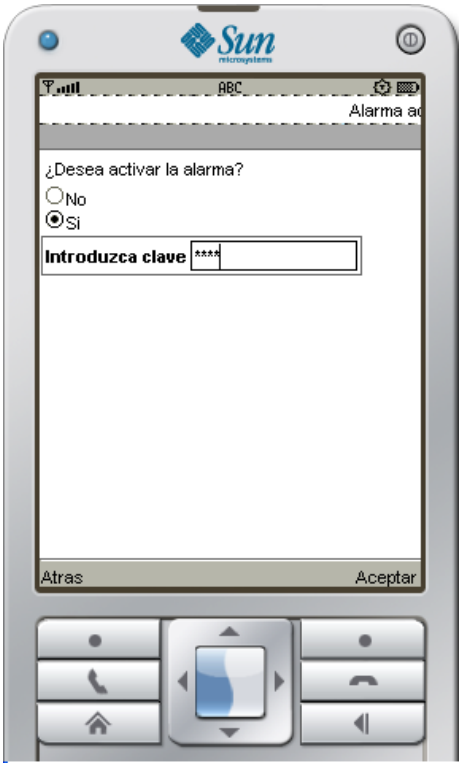


Figura 139. Introducción de clave para activar la alarma

Figura 140. Pantalla con alarma activada (punto verde)

Figura 141. Submenú de selección de Temperatura.

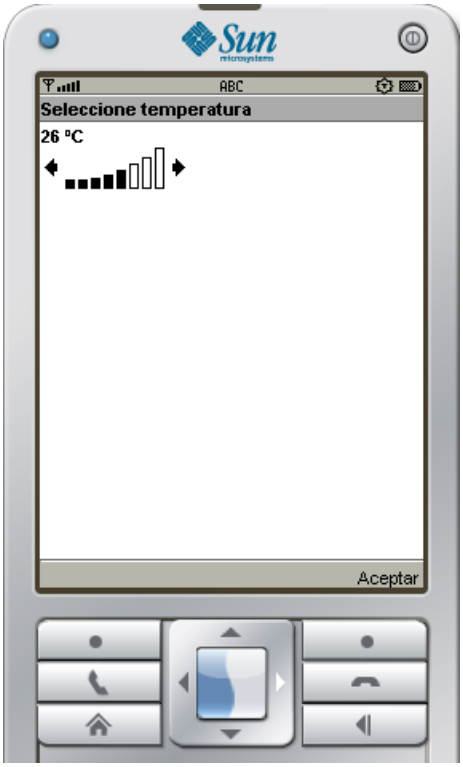


Figura 142. Selección de temperatura de una habitación.

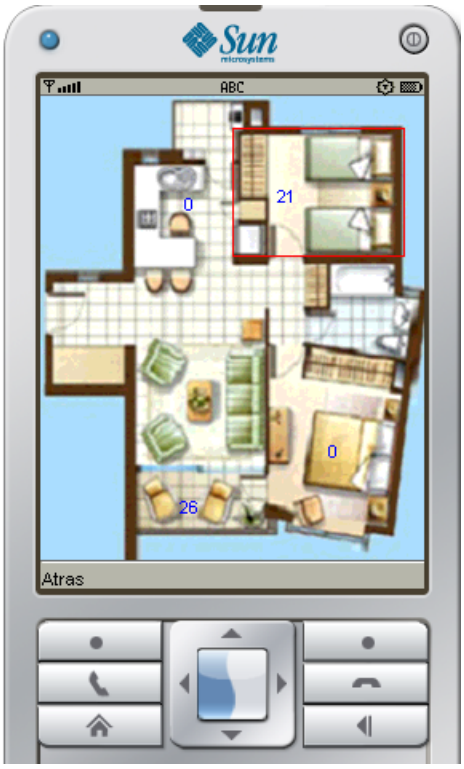


Figura 143. Pantalla con dos temperaturas seleccionadas

Figura 144. Submenú de activación de luces

Figura 145. Pantalla con 3 luces activadas

Figura 146. Submenú Video (eligiendo canal)

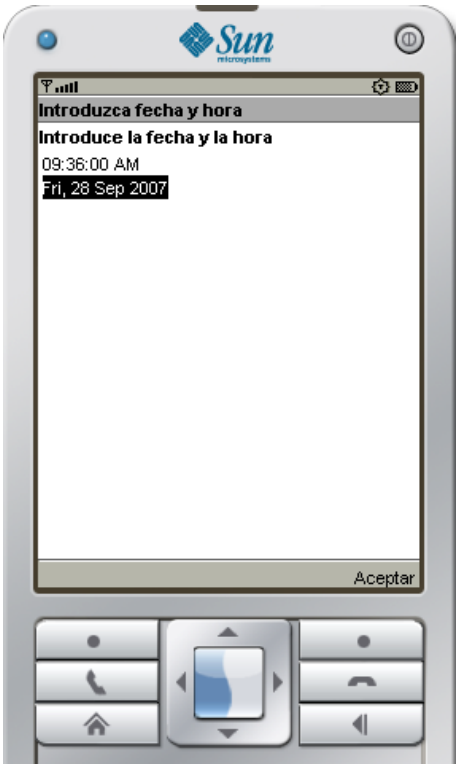


Figura 147. Menú de programación del Video

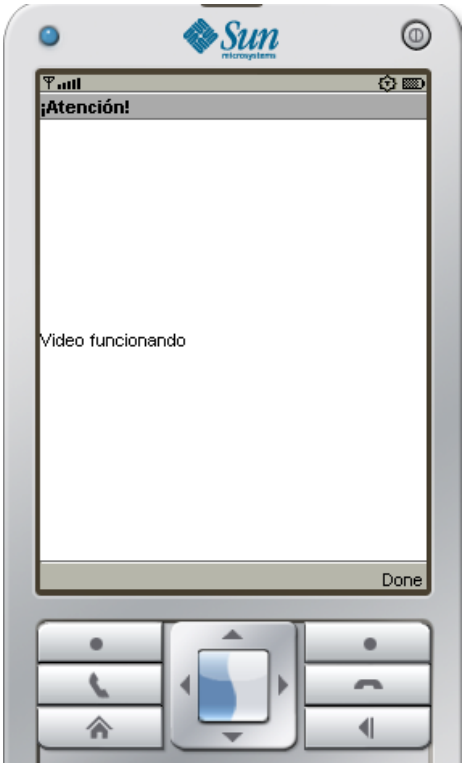


Figura 148. Pantalla emergente al pulsar Video después de haberlo programado.

Figura 149. Submenú Microondas

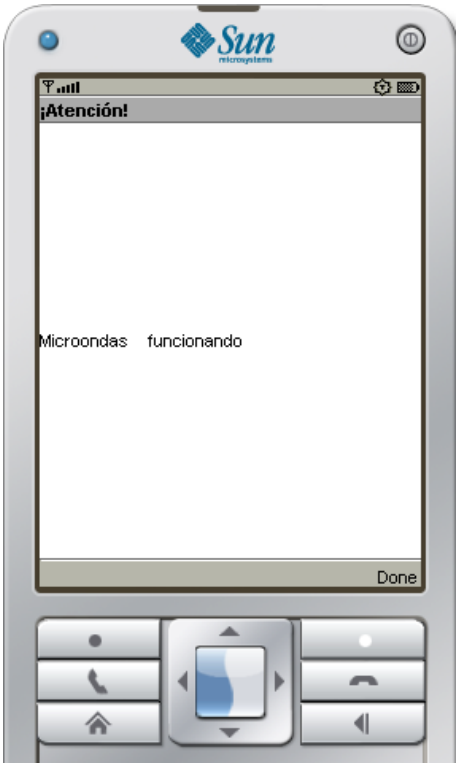


Figura 150. Pantalla emergente al pulsar sobre el submenú Microondas estando este funcionando.

Capitulo 4

C++

4.1 Introducción a C++

C++ es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C.

Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes como ROOT (enlace externo). Las principales características del C++ son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica (*templates*).

Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- Posibilidad de redefinir los operadores (sobrecarga de operadores)
- Identificación de tipos en tiempo de ejecución (*RTTI*)

C++ está considerado por muchos como el lenguaje más potente, debido a que permite trabajar tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que "dificulta" mucho su aprendizaje.

El nombre **C++** fue propuesto por Rick Masciatti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, "C++" significa "incremento de C" y se refiere a que C++ es una extensión de C.

4.2 Conceptos generales de la programación orientada a objetos

- *Clase*: Es una plantilla que define la estructura de un conjunto de objetos, que al ser creados se llamarán las instancias de la clase. Esta estructura está compuesta por la definición de los atributos y la implementación de las operaciones (métodos).
- *Objeto*: Es la implementación de una instancia de clase, es decir, una ocurrencia de esta, que tiene los atributos definidos por la clase, y sobre la que se puede ejecutar las operaciones definidas en ella.
- *Identidad*: Característica de cada objeto que lo diferencia de los demás, incluyendo de aquellos que pudieran pertenecer a la misma clase y tener los mismos valores en sus atributos.
- *Herencia*: Es la capacidad que tienen las clases para heredar propiedades y métodos de otras clases.

4.2.1 Principios

Todo programa en C++ debe tener la función main() (a no ser que se especifique en tiempo de compilación otro punto de entrada, que en realidad es la función que tiene el main():

La función main debe tener uno de los siguientes prototipos:

- int main()
- int main(int argc, char** argv)

La primera es la forma por defecto de un programa que no recibe parámetros ni argumentos. La segunda forma tiene dos parámetros: *argc*, un número describiendo el número de argumentos del programa (incluyendo el nombre del programa mismo), y *argv*, un puntero a un array de punteros, de *argc* elementos, donde el elemento argv[i] representa el *i*-ésimo argumento entregado al programa.

El tipo de retorno de main es **int**. Al finalizar la función main, debe incluirse el valor de retorno (por ejemplo, return 0; aunque el estándar prevé solamente dos posibles valores de retorno: EXIT_SUCCESS y EXIT_ERROR, definidas en el archivo cstddef), o salir por medio de la función exit. Alternativamente puede dejarse en blanco, en cuyo caso el compilador es responsable de agregar la salida adecuada.

4.2.2 El Concepto de Clase

Los objetos en C++ son abstraídos mediante una Clase. Según el paradigma de la programación orientada a objetos un objeto consta de:

1. Métodos o funciones
2. Atributos o Variables Miembro

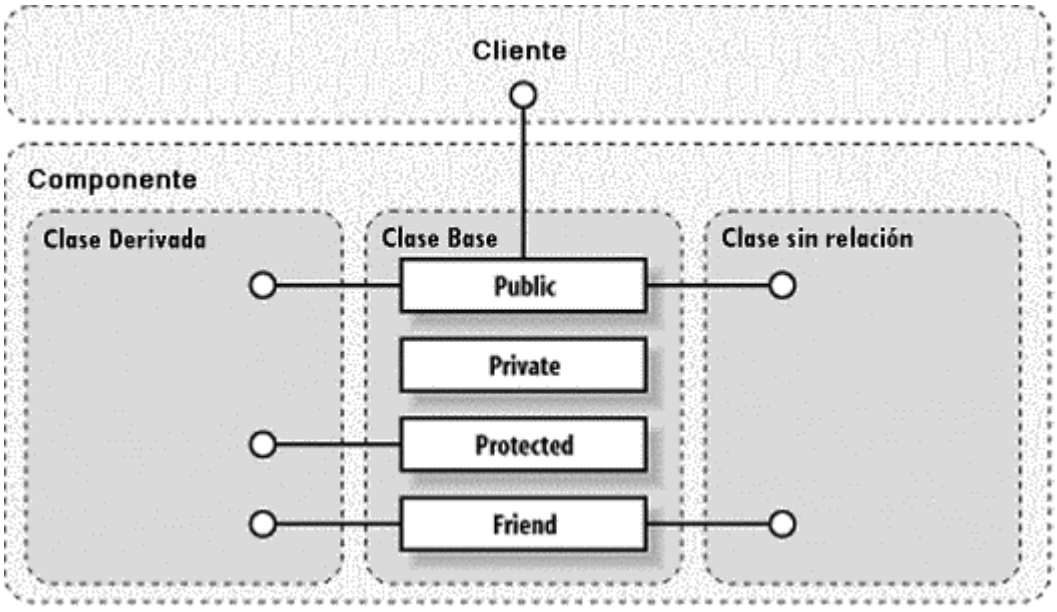


Figura 151. Diagrama de concepto de clase

4.2.3 Constructores

Son unos métodos especiales que se ejecutan automáticamente al crear un objeto de la clase. En su declaración no se especifica el tipo de dato que devuelven, y poseen el mismo nombre que la clase a la que pertenecen. Al igual que otros métodos, puede haber varios constructores sobrecargados, aunque no pueden existir constructores virtuales.

Como característica especial a la hora de implementar un constructor, justo después de la declaración de los parámetros, se encuentra lo que se llama "lista de inicializadores". Su objetivo es llamar a los constructores de los atributos que conforman el objeto a construir.

4.2.4 Destructores

Los destructores son funciones miembro especiales llamadas automáticamente en la ejecución del programa, y que por tanto no deben ser llamadas explícitamente por el programador. Su cometido es liberar los recursos computacionales que el objeto de dicha clase haya adquirido en tiempo de ejecución al expirar este.

Los destructores son invocados automáticamente al alcanzar el flujo del programa el fin del ámbito en el que está declarado el objeto.

Existen dos tipos de destructores pueden ser públicos o privados, según si se declaran:

- si es público se llama desde cualquier parte del programa para destruir el objeto.
- si es privado no se permite la destrucción del objeto por el usuario.

4.2.5 Funciones Miembro

Función miembro es aquella que está declarada en ámbito de clase. Son similares a las funciones habituales, con la salvedad de que el compilador realizara el proceso de Decoración de nombre (*Name Mangling*). Cambiará el nombre de la función añadiendo un identificador de la clase en la que está declarada, pudiendo incluir caracteres especiales o identificadores numéricos. Además, las funciones miembro reciben implícitamente un parámetro adicional, el puntero this, que referencia al objeto que ejecuta la función.

Las funciones miembro se invocan accediendo primero al objeto al cual se refieren, con la sintaxis: myobject.mymemberfunction(), esto es un claro ejemplo de una función miembro.

4.2.6 Plantillas

Las plantillas son el mecanismo de C++ para implantar el paradigma de la programación genérica. Permiten que una clase o función trabaje con tipos de datos abstractos, especificándose más adelante cuales son los que se quieren usar. Por ejemplo, es posible construir un vector genérico que pueda contener cualquier tipo de estructura de datos. De esta forma se pueden declarar objetos de la clase de este vector que contengan enteros, flotantes, polígonos, figuras, fichas de personal, etc.

La declaración de una plantilla se realiza anteponiendo la declaración template <typename A, ...> a la declaración de la estructura (clase, estructura o función) deseado.

4.2.7 Clases Abstractas

En C++ es posible definir clases abstractas. Una clase abstracta, o clase base abstracta (ABC), es una que está diseñada sólo como clase *padre* de las cuales se deben derivar clases hijas. Una clase abstracta se usa para representar aquellas entidades o métodos que después se implementarán en las clases derivadas, pero la clase abstracta en sí no contiene ninguna implementación, solamente representa los métodos que se deben implementar. Por ello, no es posible instanciar una clase abstracta, pero sí una clase concreta que implemente los métodos definidos en ella.

Las clases abstractas son útiles para definir interfaces, es decir, un conjunto de métodos que definen el comportamiento de un módulo determinado. Estas definiciones pueden utilizarse sin tener en cuenta la implementación que se hará de ellos.

En C++ los métodos de las clases abstractas se definen como funciones virtuales puras.

4.2.8 Espacios de Nombres

Una adición a las características de C son los espacios de nombre (*namespace* en inglés), los cuales pueden describirse como áreas virtuales bajo las cuales ciertos nombres de variable o tipos tienen validez. Esto permite evitar las ocurrencias de conflictos entre nombres de funciones, variables o clases.

El ejemplo más conocido en C++ es el espacio de nombres std, el cual almacena todas las definiciones nuevas en C++ que difieren de C (algunas estructuras y funciones), así como las funcionalidades propias de C++ (streams) y los componentes de la biblioteca STL.

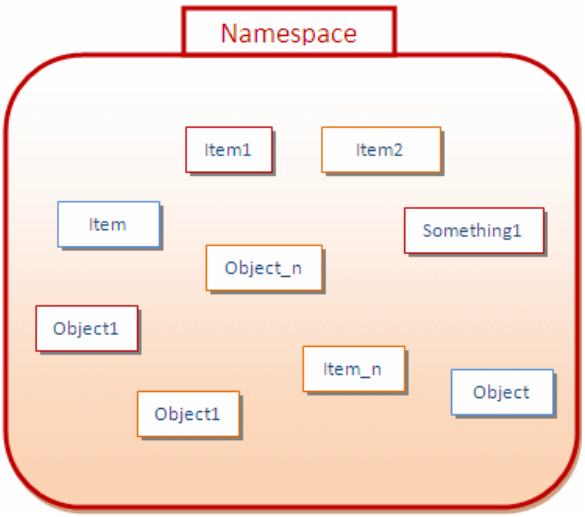


Figura 152. Ilustración de Namespace

4.2.9 Excepciones

C++ permite la existencia de *excepciones*, las cuales son una metodología de flujo de ejecución basada en la prueba del código deseado (try) seguida por la intercepción de ciertas condiciones bajo un flujo de programa adicional (catch). La declaración de estas condiciones se hace "arrojando" (throw) sentencias especiales que son capturadas por el flujo catch correspondiente.

Es buena idea al crear nuevas excepciones derivarlas de std::exception ya que es el bloque *catch* que muchos programadores colocan por defecto.

Si una excepción se propaga sin ser atrapada por un bloque *catch*, y llegara hasta el punto de terminación del programa, se produce la terminación abrupta de éste ("abort").

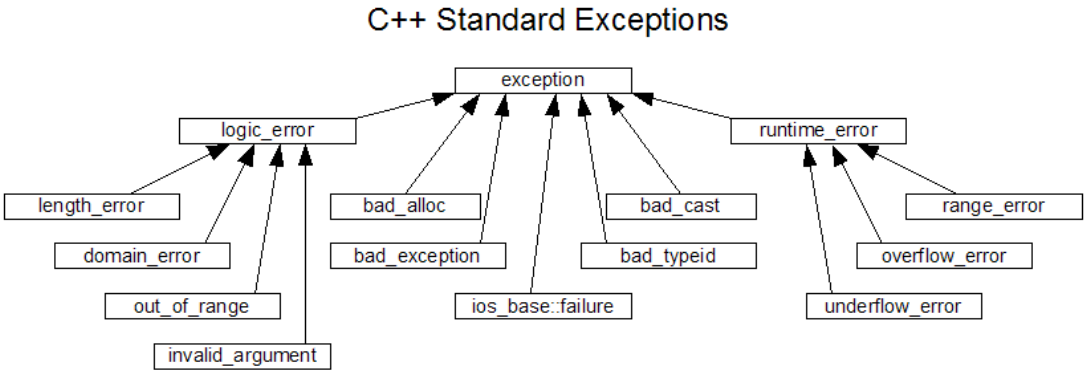


Figura 153. Diagrama de las Excepciones en C++

4.2.10 Herencia

Existen varios tipos de herencia entre clases en el lenguaje de programación C++. Estas son:

4.2.10.1 Herencia Simple

La herencia en C++ es un mecanismo de abstracción creado para poder facilitar, y mejorar el diseño de las clases de un programa. Con ella se pueden crear nuevas clases a partir de clases ya hechas, siempre y cuando tengan un tipo de relación especial.

En la herencia, las clases derivadas "heredan" los datos y las funciones miembro de las clases base, pudiendo las clases derivadas redefinir estos comportamientos (polimorfismo) y añadir comportamientos nuevos propios de las clases derivadas. Para no romper el principio de encapsulamiento (ocultar datos cuyo conocimiento no es necesario para el uso de las clases), se proporciona un nuevo modo de visibilidad de los datos/funciones: "protected". Cualquier cosa que tenga visibilidad protected se comportará como pública en la clase Base y en las que componen la jerarquía de herencia, y como privada en las clases que NO sean de la jerarquía de la herencia.

Antes de utilizar la herencia, tenemos que hacer una pregunta, y si tiene sentido, se puede intentar usar esta jerarquía: Si la frase <claseB> ES-UN <claseA> tiene sentido, entonces estamos ante un posible caso de herencia donde clase A será la clase base y clase B la derivada.

Por último, hay que mencionar que existen 3 clases de herencia que se diferencian en el modo de manejar la visibilidad de los componentes de la clase resultante:

- Herencia publica (class Derivada: public Base) : Con este tipo de herencia se respetan los comportamientos originales de las visibilidades de la clase Base en la clase Derivada.

- Herencia privada (clase Derivada: private Base) : Con este tipo de herencia todo componente de la clase Base, será privado en la clase Derivada (siempre será privado aunque ese dato fuese público en la clase Base)

- Herencia protegida (clase Derivada: protected Base): Con este tipo de herencia, todo componente publico y protegido de la clase Base, será protegido en la clase Derivada, y los componentes privados, siguen siendo privados.

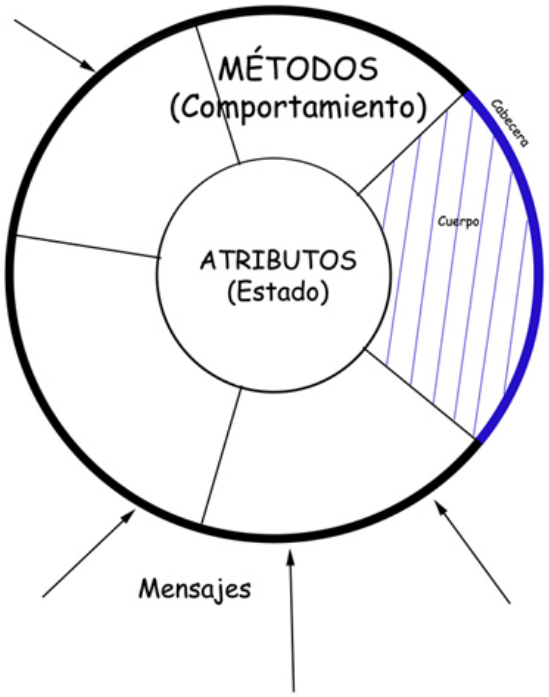


Figura 155. Ilustración de Herencia Simple

4.2.10.2 Herencia Múltiple

La herencia múltiple es el mecanismo que permite al programador hacer clases derivadas a partir, no de una sola clase base, sino de varias. Para entender esto mejor, pongamos un ejemplo: Cuando ves a quien te atiende en una tienda, como persona que es, podrás suponer que puede hablar, comer, andar, pero, por otro lado, como empleado que es, también podrás suponer que tiene un jefe, que puede cobrarte dinero por la compra, que puede devolverte el cambio, etc.

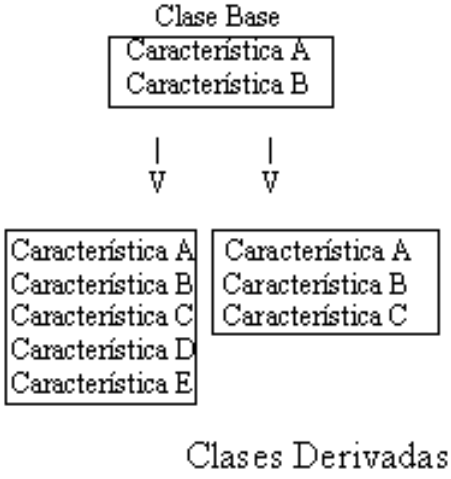


Figura 155. Diagrama de Herencia Multiple.

4.2.11 Sobrecarga de Operadores

Es posible definir el comportamiento de un operador del lenguaje para que trabaje con tipos de datos definidos por el usuario. No todos los operadores de C++ son factibles de sobrecargar, y, entre aquellos que pueden ser sobrecargados, se deben cumplir condiciones especiales. En particular, los operadores sizeof y :: no se pueden sobrecargar

No es posible en C++ crear un operador nuevo.

Los comportamientos de los operadores sobrecargados se implementan de la misma manera que una función, salvo que esta tendrá un nombre especial: *Tipo de dato de devolución* operator<token del operador> (parámetros)

Dado que estos operadores son definidos para un tipo de datos definido por el usuario, éste es libre de asignarles cualquiera semántica que desee. Sin embargo, se considera de primera importancia que las semánticas sean tan parecidas al comportamiento natural de los operadores como para que el uso de los operadores sobrecargados sea intuitivo.

4.2.12 Biblioteca Estándar de Plantillas (STL)

Los lenguajes de programación suelen tener una serie de bibliotecas integradas para la manipulación de datos a nivel más básico. En C++, además de poder usar las bibliotecas de C, se puede usar la nativa STL (Standard Template Library), propia del lenguaje. Proporciona una serie de clases parametrizadas que permiten efectuar operaciones sobre el almacenado de datos, procesado y flujos de entrada/salida. La STL más que una biblioteca es un conjunto de ellas. De esta forma únicamente se incluyen en el fichero ejecutable final aquellas que sean necesarias para la aplicación que se esté programando, reduciendo drásticamente el uso innecesario de memoria.

- **ostreams / istreams**

Cabe destacar las clases basic_ostream y basic_stream, y los objetos cout y cin, pertenecientes a estas clases, respectivamente. Proporcionan la entrada y salida estándar de datos (teclado/pantalla). También está disponible cerr, similar a cout, usado para la salida estándar de errores. Estas clases tienen sobrecargados los operadores << y >>, respectivamente, con el objeto de ser útiles en la inserción/extracción de datos a dichos flujos. Son operadores inteligentes, ya que son capaces de adaptarse al tipo de datos que reciben, aunque tendremos que definir el comportamiento de dicha entrada/salida para clases/tipos de datos definidos por el usuario.

Es posible formatear la entrada/salida, indicando el número de dígitos decimales a mostrar, si los textos se pasarán a minúsculas o mayúsculas, si los números recibidos están en formato octal o hexadecimal, etc.

- **fstreams**

Tipo de flujo para el manejo de ficheros. La definición previa de *ostreams/istreams* es aplicable a este apartado. Existen tres clases (ficheros de lectura, de escritura o de lectura/escritura): ifstream,ofstream y fstream.

Para cerrar un fichero, puede usarse el método close, o esperar a que el destructor de las clases lo cierre automáticamente.

- **sstreams**

Se destacan dos clases, ostringstream e istringstream. Todo lo anteriormente dicho es aplicable a estas clases. Tratan a una cadena como si de un flujo de datos se tratase. ostringstream permite elaborar una cadena de texto insertando datos cual flujo, e istringstream puede extraer la información contenida en una cadena (pasada como parámetro en su constructor) con el operador >>.

4.2.13 Contenedores

Son clases plantillas especiales utilizadas para almacenar tipos de datos genéricos, sean cuales sean. Según la naturaleza del almacenado, disponemos de varios tipos:

- Vectores: Se definen por vector<tipo_de_dato> nombre_del_vector;
- Equivalen a los array de cualquier lenguaje, con diversas salvedades. Tienen tamaño dinámico, con lo que se puede insertar elementos aún si el vector está lleno. A diferencia de los vectores clásicos a bajo nivel de C, también pueden lanzar excepciones si se accede a un elemento cuyo rango está fuera del vector en cuestión, usando, en vez del operador [], el método *at()*.
- Colas dobles: son parecidas a los vectores, pero tienen mejor eficiencia para agregar o eliminar elementos en las "puntas".
- Listas.
- Adaptadores de secuencia.
- Contenedores asociativos: map y multimap, que permiten asociar una "clave" con un "valor".
- Contenedores asociativos: set y multiset, que ofrecen solamente la condición de "pertenencia", sin la necesidad de garantizar un ordenamiento particular de los elementos que contienen.

4.2.14 Iteradores

Pueden considerarse como una generalización de la clase de "puntero". Un iterador es un tipo de dato que permite el recorrido y la búsqueda de elementos en los contenedores. Como las estructuras de datos (contenedores) son clases genéricas, y los operadores (algoritmos) que deben operar sobre ellas son también genéricos (funciones genéricas), Stepanov y sus colaboradores tuvieron que desarrollar el concepto de iterador como elemento o nexo de conexión entre ambos. El nuevo concepto resulta ser una especie de punteros que señalan a los diversos miembros del contenedor (punteros genéricos que como tales no existen en el lenguaje).

4.2.15 Algoritmos

Combinando la utilización de templates y un estilo específico para denotar tipos y variables, la STL ofrece una serie de funciones que representan operaciones comunes, y cuyo objetivo es "parametrizar" las operaciones en que estas funciones se ven involucradas de modo que su lectura, comprensión y mantenimiento, sean más fáciles de realizar.

Un ejemplo es la función copy, la cual simplemente copia variables desde un lugar a otro. Más estrictamente, copia los contenidos cuyas ubicaciones están delimitadas por dos iteradores, al espacio indicado por un tercer iterador. La sintaxis es:

copy (inicio_origen, fin_origen, inicio_destino);

De este modo, todos los datos que están entre inicio_origen e fin_origen, exclusive el dato ubicado en este último, son copiados a un lugar descrito o apuntado por inicio_destino.

Entre las funciones más conocidas están swap (variable1, variable2), que simplemente intercambia los valores de variable1 y variable2; max (variable1, variable2) y su símil min (variable1, variable2), que retornan el máximo o mínimo entre dos valores; find (inicio, fin, valor) que busca valor en el espacio de variables entre inicio y fin; etcétera.

Los algoritmos son muy variados, algunos incluso tienen versiones específicas para operar con ciertos iteradores o contenedores, y proveen un nivel de abstracción extra que permite obtener un código más "limpio", que "describe" lo que se está haciendo, en vez de hacerlo paso a paso explícitamente.

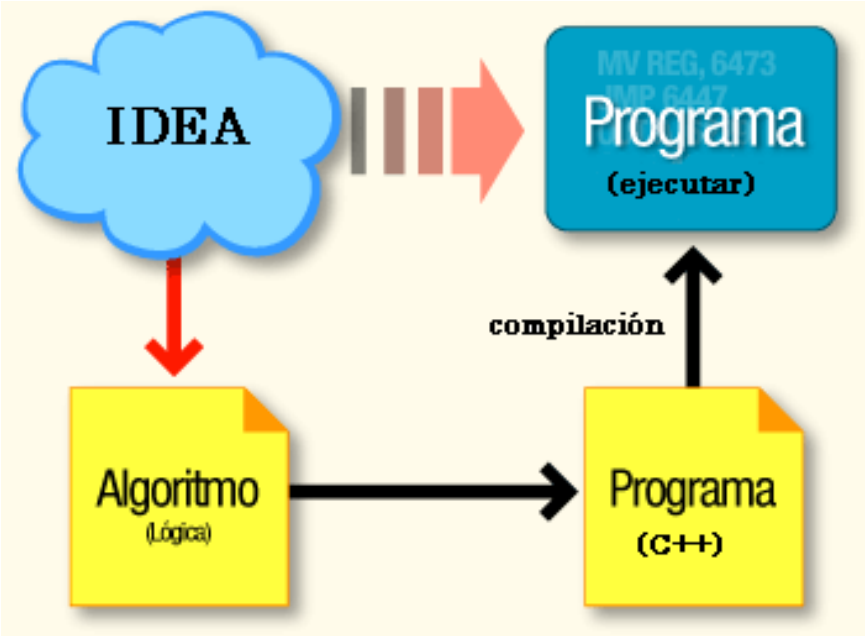


Figura 156. Ilustración de lógica algorítmica e implementación

4.3 C++ en Symbian

Como se menciono en capítulos anteriores uno de los lenguajes en los que se pueden realizar aplicaciones Symbian es C++. Este es un lenguaje muy potente proveniente de C pero con una orientación clara a objetos. Este lenguaje es algo mas rápido en el momento de compilación que Java y con el se consigue un mejor aprovechamiento de los recursos y capacidades del sistema operativo Symbian. Contiene librerías para manejar cada uno de los aspectos de Symbian, algo que no ocurría en Java, ya que Java no podía manejar ciertas librerías. En este caso, con este lenguaje todo es accesible.

Por otro lado, C++ es un lenguaje algo mas complejo que Java, su uso de punteros y su manera de generar código obliga al programador a tener que implementarlo casi todo. En Java con la utilización de su multitud de librerías la tarea de crear una nueva aplicación era más rápida ya que se produce un mayor aprovechamiento de las mismas. En este caso con C++ el programador se vera obligado a crearlo casi todo y a controlar todos los eventos producidos, además de tener que liberar los recursos después de haberlos utilizado, controlar el uso de las pila, etc.

En definitiva, C++ sobre Symbian OS es mas potente que Java en cuanto a capacidad de manejo de recursos y accesibilidad, pero en contra tiene que la realización de nuevas aplicaciones es mas tediosa y costosa que en el anterior lenguaje visto en este proyecto.

En la siguiente figura se muestra como se produce la creación de un archivo .sis instalable en el dispositivo y las operaciones que se realizan posteriormente a la creación del sis para firmar la aplicación. Con esta operación de firma de aplicaciones Symbian intenta proteger su sistema de virus, haciendo que todas las aplicaciones pasen por un ente público que se encargue de comprobar su autenticidad y su contenido para posteriormente firmar la misma en caso de que todo sea correcto. El ente que firma las aplicaciones se llama Symbian Signed y su página web desde la cual se pueden firmar aplicaciones es la siguiente: *https://www.symbiansigned.com/app/page*

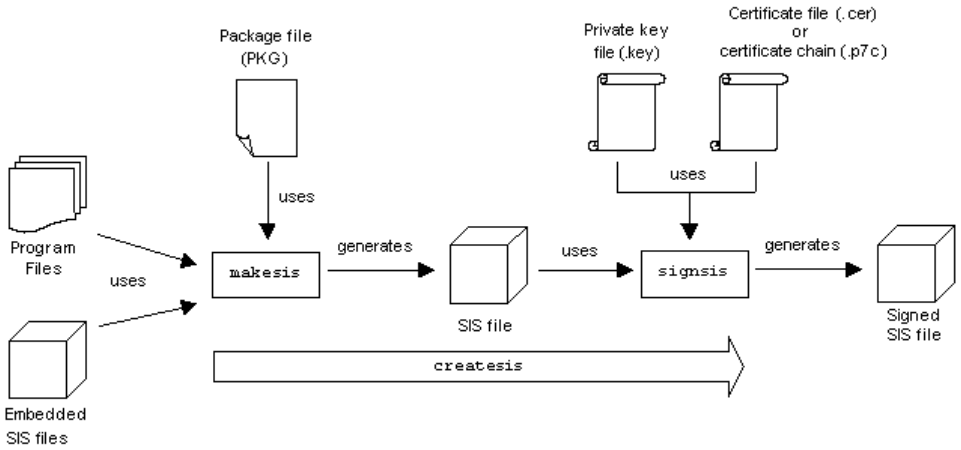


Figura 157. Diagrama de creación y firma de un archivo SIS.

Solamente notar como última apreciación que solo pueden firmar aplicaciones aquellas personas que estén dadas de alta en la web. Para poder darte de alta Symbian Signed posee unas listas en las que organiza las cuentas de e-mail que se le pasan, si tu cuenta es publica como Hotmail, gmail, yahoo, etc. No te dejará darte de alta porque considera estos emails como peligros potenciales. Es necesaria una cuenta de empresa.

4.4 Desarrollo de aplicaciones y uso de Herramientas

4.4.1 Introducción a herramientas y entorno Symbian

4.4.1.1 Introducción

Como siempre a la hora de empezar con un nuevo lenguaje en una nueva plataforma es necesario comenzar comprendiendo las librerías más básicas y familiarizándose con el entorno en el cual se van a realizar las aplicaciones. Por ello, en esta primera parte se va realizar un estudio del entorno que se va a utilizar y de las herramientas que facilita el mismo. Entre estas herramientas se puede destacar además del propio uso del IDE elegido, se explicaran el emulador, el debugador y se realizará un primer proyecto implementando la típica aplicación Hola Mundo. Se explicaran en el mismo cada una de las librerías utilizadas y cada parte del programa en profundidad. Aplicándole la relevancia adecuada a cada una de ellas. También se verán los modos de compilación posible, los archivos que componen cada proyecto y sus respectivas ubicaciones dentro del entorno.

Como se vio en la parte de J2ME, todo lo que a continuación se abordara será para la plataforma UIQ bajo el sistema operativo Symbian, pero en este caso el lenguaje elegido para implementar las aplicaciones será C++ en lugar de Java. La herramienta que se utilizará será Carbide C++ la cual es propiedad de Nokia.

4.4.1.2 Descarga e instalación de las herramientas necesarias.

Carbide C++ es una herramienta perteneciente a Nokia, la cual va a ser utilizada en este apartado del proyecto para realizar las aplicaciones, compilarlas, debuguearlas, emularlas y generar los archivos de instalación necesarios (.sis) para el terminal. Este IDE no es freeware, pero desde la página web de fórum Nokia se puede descargar una versión de evaluación para probar dicho entorno durante 21 días. Tras los cuales podrá adquirirse una licencia del programa completo. La página de la aplicación en concreto es la siguiente:

http://www.forum.nokia.com/main/resources/tools_and_sdks/carbide/index.html

También deberán descargarse los SDK necesarios para el terminal donde se quiera instalar la aplicación. En este caso será necesario un SDK para UIQ, en concreto para su versión 2.0 la cual es la que usa el Sony Ericsson P800. La página web donde se pueden encontrar estos SDK es la siguiente:

http://developer.sonyericsson.com/site/global/docstools/symbian/p_symbian.jsp

En esa misma página se podrá descargar una actualización específica para el terminal (P800) que modifica el interfaz del emulador, poniendo como pantalla de prueba el mismo terminal que utilizamos, lo que proporciona una visión totalmente exacta del resultado de la aplicación elaborada.

Para la instalación de los mismos los pasos a seguir son los siguientes:

1. Instalación del Carbide C++. Tiene una instalación guiada normal y corriente, Se hace doble click en el archivo que se ha descargado y solo se debe elegir el directorio donde se quieres instalar la aplicación. Es bueno elegir un directorio diferente del que te sale por defecto, para tener controlado el lugar donde se realiza la instalación ya que por ejemplo habrá que seleccionar un directorio de trabajo (work) donde se guardaran todos los proyectos realizados. Este directorio no puede contener espacios en su nombre porque si no el SDK no compilará.

2. Instalación de los SDKs. Se pueden instalar todos los que se quieran e incluso en la misma unidad de disco, pero no pueden estar en el directorio raíz. Durante el proceso de instalación se le pregunta al usuario donde se quiere tener el directorio raíz. La estructura del directorio se distribuye entonces de tal manera que se pueden agrupar distintos SDKs juntos en ese directorio raíz.

3. Instalación de actualización para Sony Ericsson P800. Simplemente se deberá de hacer doble click sobre el archivo descargado y ella sola se instala en su lugar correspondiente.

Figura 158. Fotografía del Sony Ericsson P800 ejecutando una aplicación.

4.4.1.3 Creación de un proyecto con Carbide C++

Existen varias maneras de crear un proyecto con Carbide C++. Todas ellas iguales en resultado pero el punto de inicio es diferente:

1. Creación de proyecto nuevo: Lo primero que se va a ver es la creación de un proyecto de la nada. Este es el caso en el que no se tiene nada para empezar en el proyecto y se pretende crearlo desde el inicio. Para ello se deberá abrir Carbide C++ y seleccionar en la pestaña de arriba a la izquierda llamada File -> New -> Project, lo cual producirá una ventana como la siguiente imagen.

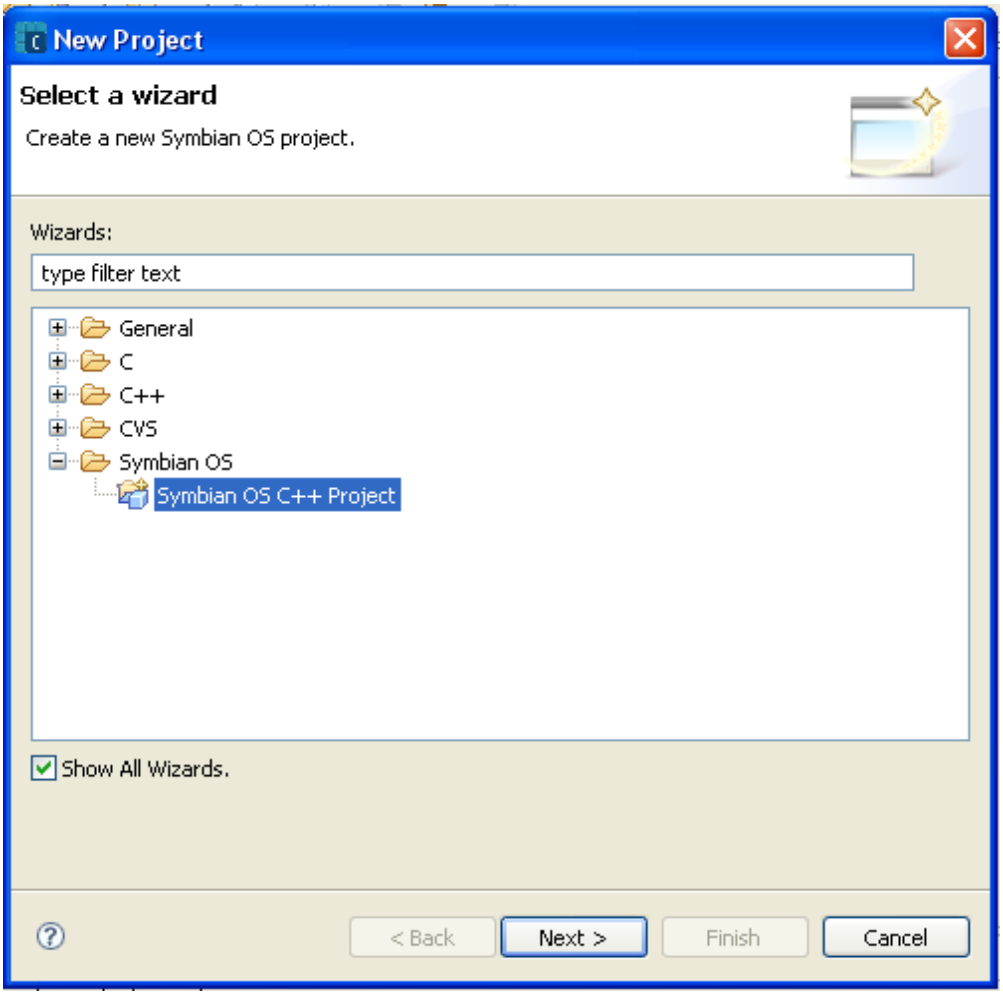


Figura 159. Pantalla 1 de creación de un proyecto nuevo con Carbide C++

En esta ventana se deberá elegir la opción de proyecto Symbian OS C++ proyect y pulsar Next. Una vez hecho esto, el wizard mostrara una nueva pantalla destinada a la elección del tipo de proyecto concreto para el cual se quiere realizar la aplicación. Dara a elegir entre multitud de opciones, entre diferentes plataformas (s60, s80, UIQ). Pero la elección en esta ocasión será un proyecto genérico de Symbian OS, en concreto, una aplicación de consola Básica. Ya que se pretende crear un Hola Mundo.

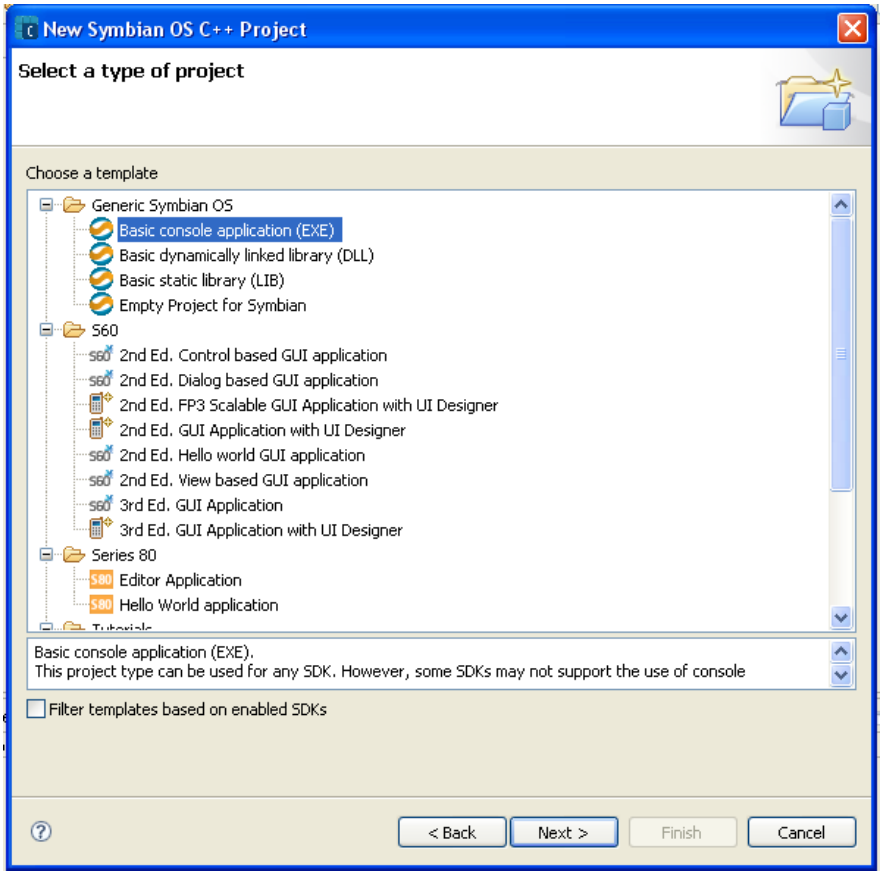


Figura 160. Pantalla 2 de creación de un proyecto nuevo con Carbide C++

En la siguiente ventana se deberá elegir el nombre del proyecto que se pretende crear y la ubicación del mismo. Es decir el directorio donde se guardará.

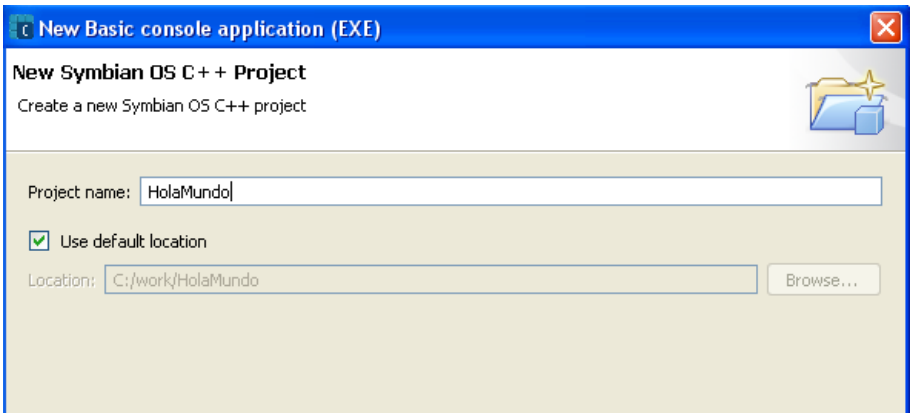


Figura 161. Pantalla 3 de creación de un proyecto nuevo con Carbide C++

Una vez elegido el nombre y el directorio se pulsará Next lo cual abrirá la siguiente ventana. En esta siguiente ventana se deberá elegir el SDK en concreto para el cual se va a querer compilar la aplicación (generar su respectivo .sis). En esta ventana aparecerán todos los SDKs que se hayan instalado en el ordenador. En este caso aparecen tres, dos configuraciones para el 2.0 y una configuración para el 2.1. Se elegirá en este caso la configuración del 2.1.

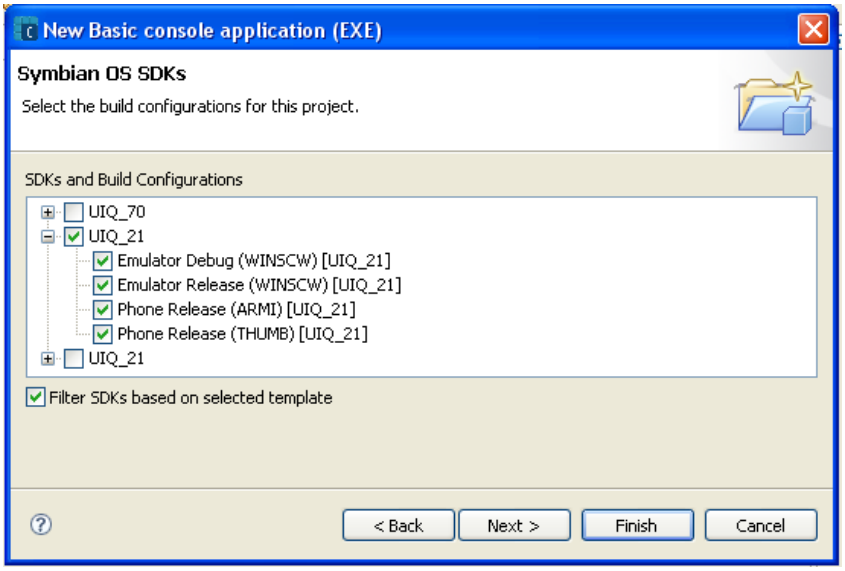


Figura 162. Pantalla 4 de creación de un proyecto nuevo con Carbide C++

En la siguiente ventana Carbide C++ generará un UID para la aplicación que se esta creando. Este UID lo escogerá aleatoriamente del conjunto de UIDs de libre distribución, es decir, de los designados para pruebas. Si se quisiera realizar la distribución de una aplicación de manera comercial se debería solicitar un UID de comercialización a UIQ. En la siguiente ventana también se escogerá el nombre de la persona que esta creando la aplicación y hay un espacio destinado a un breve comentario de la misma, o unas notas de Copyright del autor.

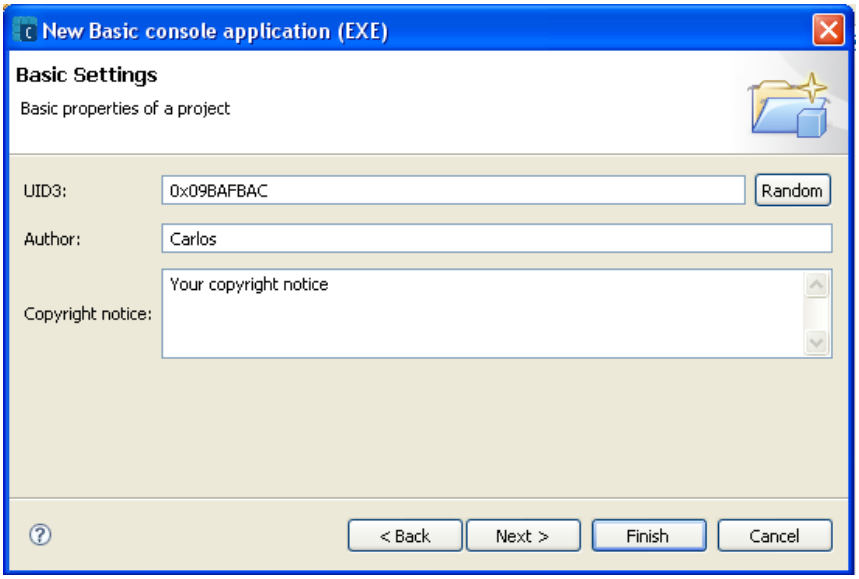


Figura 163. Pantalla 5 de creación de un proyecto nuevo con Carbide C++

Por último la siguiente ventana es de confirmación para los directorios que se crearan con el proyecto. En ella habrán de confirmarse los mismos, según los que se vayan a necesitar. Y se estará en disposición de pulsar Finish, tras lo cual se generará automáticamente un proyecto según los parámetros elegidos.

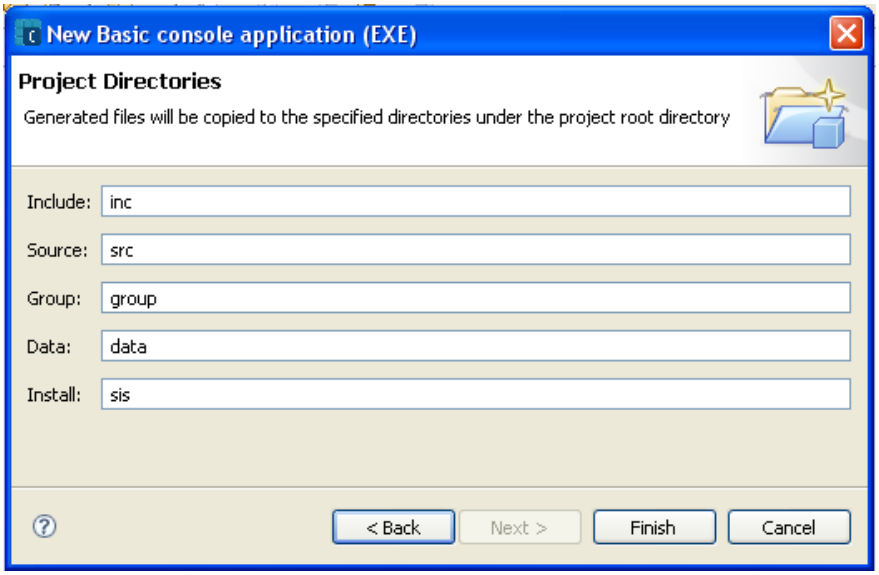


Figura 164. Pantalla 6 de creación de un proyecto nuevo con Carbide C++

2. Creación de proyecto a partir de archivo mmp: En este punto se creará un proyecto a partir de su archivo de especificación mmp. Para ello se habrá de pinchar de nuevo en el desplegable de la parte superior izquierda llamado File y a continuación seleccionar la opción Import.

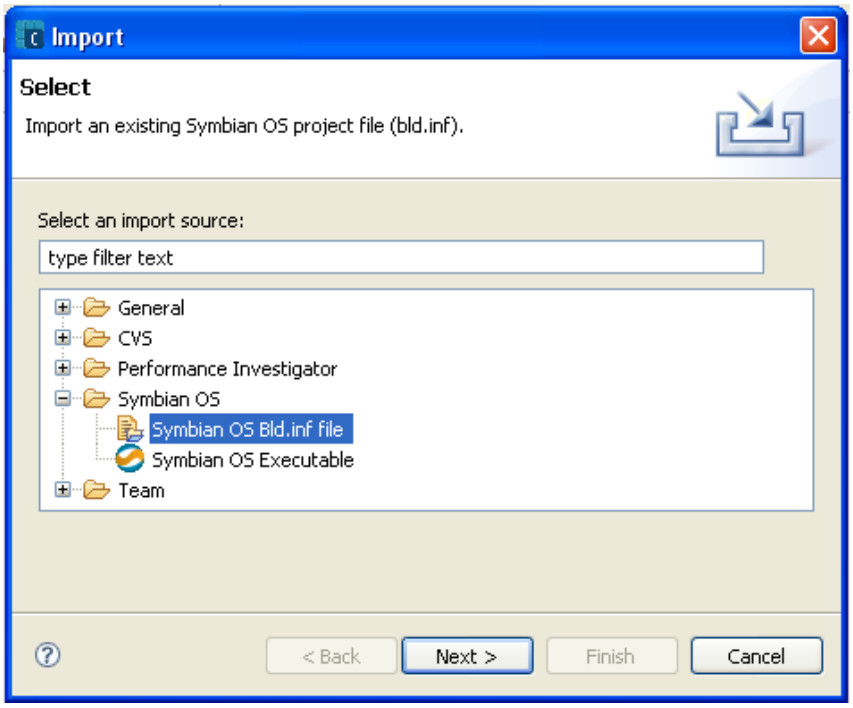


Figura 165. Pantalla 1 de creación de un proyecto a partir de archivo mmp.

Una vez hecho esto se deberá buscar la ubicación del archivo y como en el caso anterior elegir el SDK para el cual se quiere crear el proyecto, para ello simplemente se deberá ir completando las ventanas que van saliendo que son similares a las del caso anterior. Después de esto veremos una ventana para seleccionar el .mmp a partir del cual se creará el proyecto. A continuación una figura con la ventana en cuestión.

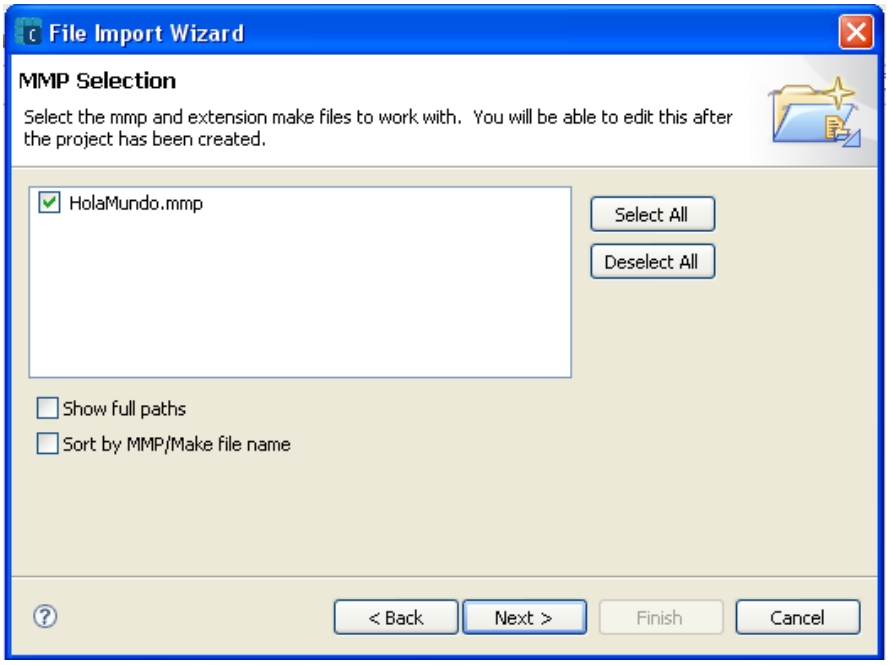


Figura 166. Pantalla 2 de creación de un proyecto a partir de archivo mmp.

Una vez hecho todo esto se pulsará Next, la siguiente ventana es de información simplemente de la ubicación del mmp, en la cual hay que pulsar Finish y el proyecto será creado. Tras lo cual el entorno tiene este aspecto.

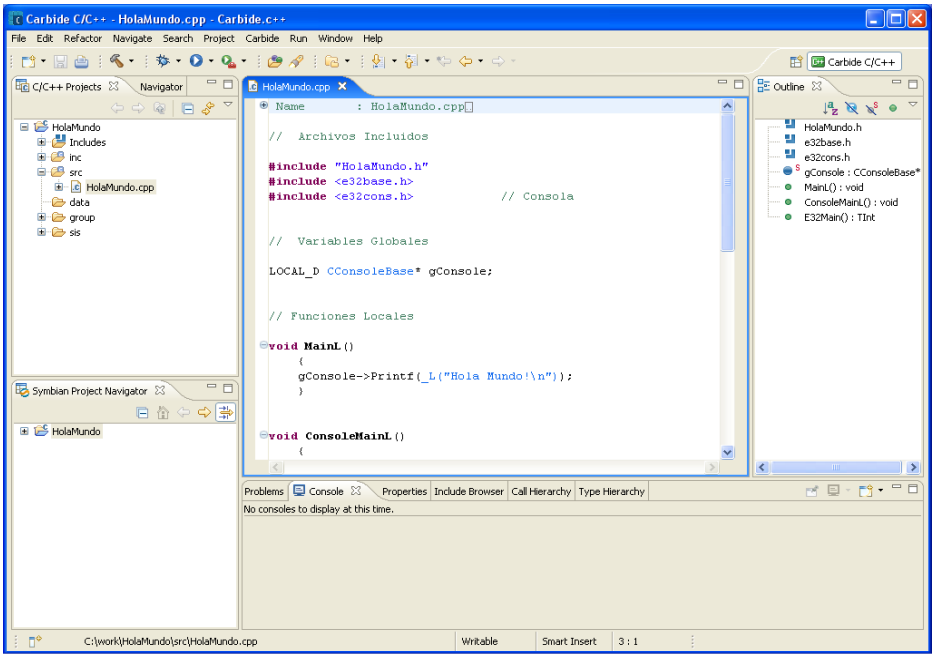


Figura 167. Aspecto del entorno Carbide C++.

4.4.1.4 Implementación de la aplicación Hola Mundo.

Antes ya se comento que en esta primera parte de utilización de Symbian OS con el lenguaje C++ lo primero que se haría seria crear un proyecto con la implementación de una aplicación básica por la cual suele empezar la introducción en una nueva plataforma. Esta aplicación es Hola Mundo, en este punto se abordará su implementación explicando las partes básicas y librerías que utiliza el programa.

Esta aplicación se compondrá de una interfaz muy simple de texto con la frase "Hola Mundo" en pantalla. El código del programa es el siguiente:

```
#include "HolaMundo.h"
#include <e32base.h>
#include <e32cons.h>     // Consola

//  Variables Globales

LOCAL_D CConsoleBase* gConsole;

// Funciones Locales

void MainL()
 {
 gConsole->Printf(_L("Hola Mundo!\n"));
 }

void ConsoleMainL()
 {

 gConsole = Console::NewL(_L("Hola Texto"),TSize(KConsFullScreen,
 KConsFullScreen));
 CleanupStack::PushL(gConsole);

 //Llamada a la funcion
 MainL();

 //Tiempo de terminacion
 User::After(5000000);   //5 segundos

 //Terminacion de la consola
 CleanupStack::PopAndDestroy(gConsole);
 }

//  Funciones Globales

GLDEF_C TInt E32Main()
 {
 // Libero recursos, la pila
 __UHEAP_MARK;
 CTrapCleanup* cleanupStack = CTrapCleanup::New();

 // Creación de la consola de salida
 TRAPD(error, ConsoleMainL());
 __ASSERT_ALWAYS(!error, User::Panic(_L("SCMP"),error));
 delete cleanupStack;
 __UHEAP_MARKEND;
 return 0;
 }
```

En esta aplicación se han utilizado tres funciones básicas:

- *MainL(),* es bajo la cual funciona toda la aplicación Hola Mundo.
- *ConsoleMainL(),* se encarga de la creación de la consola y de llamar a MainL().
- *E32Main(),* se encarga de liberar los recursos.

Para entender bien el funcionamiento de la aplicación se intentará explicar claramente que hace cada parte del programa y cual es el resultado que aporta.

El trabajo real de la aplicación se realiza en la función MainL(), esta contiene una llamada a printf() que pertenece a la clase CConsoleBase y que se encarga de mostrar en pantalla el código pasado como parámetro a ese método.

Se puede observar un símbolo que también se le pasa al método como parámetro que es este _L. Este símbolo es un macro de estilo para los descriptores Symbian.

El sistema operativo Symbian siempre comienza los programas de texto con la función E32Main(). Esta función y ConsoleMainL() construyen las dos piezas importantes de la estructura de MainL(): el limpiado de la pila y la consola respectivamente.

La declaración de E32Main() indica que esta es una función global y como se puede ver devuelve un valor Tint. Este es similar a int pero desde los compiladores C++ no pueden garantizar que este int sea un entero de 32 bits con signo ya que Symbian OS utiliza typedefs en los tipos estandar para estar seguro de que sea la implementación que sea y el compilador que se utilice la salida no varíe. Todo esto viene desprendiéndose de la necesidad de portabilidad de las aplicaciones Symbian, que a veces no se puede asegurar con total exactitud la composición de la compilación.

E32Main() se encarga de la configuración del framework de errores. Como se puede comprobar en el código, configura el limpiado de pila y después llama a ConsoleMainL() bajo el método Trap(). Este método se encarga de atrapar los errores, es como un bloque try()/catch() en Java, pero todo en uno.

La tarea principal de la función ConsoleMainL() es la de llamar a la función MainL() pero a parte de esto realiza otras tareas como:

- Crea una consola donde MainL() pondrá el texto que quiere mostrar por pantalla.
- Es la encargada del tiempo de expiración de la aplicación (en este caso 5 segundos).
- Llama a la función principal MainL().
- Destruye la consola que había creado anteriormente una vez que ya no es necesaria su utilización, liberando los recursos de esta.

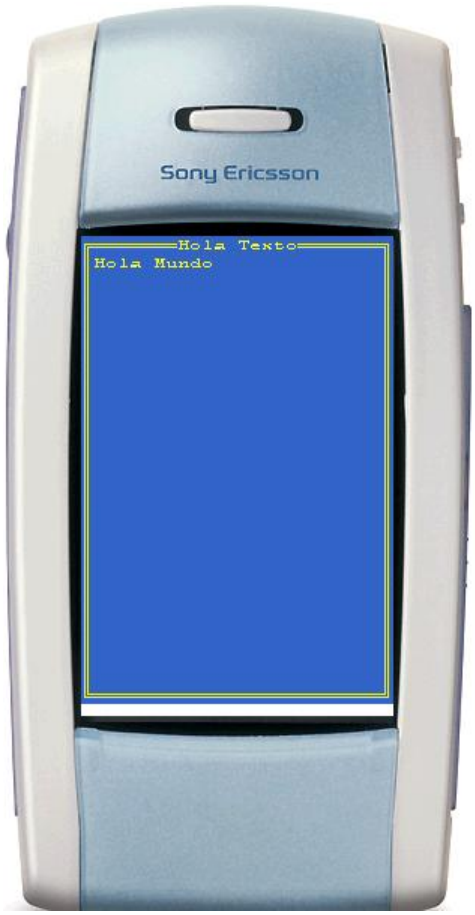


Figura 168. Emulación de la aplicación Hola Mundo sobre Carbide C++

No es necesaria la utilización del método Trap() en ConsoleMainL() al llamar a la función MainL() ya que al estar este en E32Main() desde aquí podrá capturar cualquier error que se genere en el programa.

Las cabeceras del programa e32base.h y e32cons.h son incluidas en todas las aplicaciones Symbian realizadas con C++. La primera de ellas contiene todos los métodos básicos que se pueden utilizar mientras que la segunda es utilizada para la interfaz de la consola y para los programas que funcionan en modo texto. *Figura con la emulación de Hola Mundo.*

4.4.1.5 El archivo de especificación del proyecto (mmp)

Este archivo depende del tipo de compilación que se quiera hacer. Este archivo contiene toda la información necesaria para poder compilar un archivo y generar su .pkg que será a su vez el encargado de crear el .sis que se instalará en el terminal. El mmp es utilizado para traducir todas las características del programa al compilador y variará en función de las herramientas de compilación utilizada (tipo SDK, debuguer y emulador). Como su propio nombre indica tiene la extensión .mmp y en el caso de la aplicación Hola Mundo que se ha creado tiene la siguiente forma:

```
TARGET          HolaMundo.exe
TARGETTYPE      exe
UID             0 0x0E417AE2

USERINCLUDE      ..\inc
SYSTEMINCLUDE   \epoc32\include

SOURCEPATH      ..\src
SOURCE          HolaMundo.cpp

LIBRARY          euser.lib
```

Esta compuesto por diversos campos que se van a explicar a continuación:

- TARGET/TARGETTYPE: Especifica el ejecutable que va a ser generado, y el TARGETTYPE confirma la extensión que va a tener el mismo.

- UID: Todos los programas Symbian necesitan un UID especifico, este digamos es la marca de su originalidad, sirve para distinguirlo del resto de programas y aplicaciones disponibles. Existen varios rangos de UID, en este caso Carbide utiliza un UID genérico para las aplicaciones nuevas que se crean y están sin firmar. Una vez una aplicación quiere ser comercializada necesita tener un UID especifico los cuales son otorgados por una entidad a nivel mundial.

- SOURCEPATH: Especifica la localización exacta de los archivos fuente del proyecto.

- SOURCE: Este campo le dice al compilador cuales son los archivos fuente incluidos en el proyecto, para que a la hora de compilar sepa cuales son los necesarios. Puede haber varios campos SOURCE en un archivo mmp porque un proyecto puede componerse de diversas clases.

- USERINCLUDE y SYSTEMINCLUDE: Estos dos campos muestran los directorios donde se depositan los archivos a incluir en los proyectos que no son código fuente, tales como iconos, archivos de estilo (.rss), etc.

- LIBRARY: Como su propio nombre indica especifica las librerías que deben incluirse en la aplicación para un correcto funcionamiento de la misma. En este caso (Hola Mundo) la librería incluida es euser.lib que es utilizada para todas las funciones de E32.

4.4.1.6 Compilación de una aplicación

Hay dos modos principales de compilación de una aplicación C++ sobre Symbian OS. Las posibilidades son compilación por línea de comandos y a través de Carbide C++. En este apartado se van a explicar ambas.

Compilación de una aplicación por línea de comandos

Para comenzar se deberá abrir una línea de comandos e ir al directorio en el cual se encuentran los archivos del proyecto. Esto se hace utilizando el comando cd como en la siguiente línea:

cd \work\HolaMundo

Después se deberá invocar a bldmake:

bldmake bldfiles

Una vez hecho esto se deberá comprobar los contenidos del directorio fuente en el cual habrá un nuevo archivo llamado abld.dat el cual será usado para conducir el proceso de compilación. El siguiente paso es utilizar el comando abld para arrancar el proceso completo de compilación. El comando completo es:

abld build winscw udeb

Winscw es utilizado para especificar que la compilación será para ser utilizada en el emulador. Este comando generará una lista con comandos, la cual habla de lo que se ha realizado para compilar. De la lista generada a la salida podemos extraer la conclusión de que la compilación se divide en seis fases que son las siguientes:

- La primera fase exporta copias de los archivos a sus destinos finales. Entre ellas al directorio \epoc32\include.
- En la segunda fase se crea el makefile necesario o el espacio de trabajo del IDE.
- La tercera fase importa las librerías.
- Esta cuarta fase es la de tratamiento de recursos, en ella se crean los recursos que la aplicación va a necesitar, los mapas de bits y los archivos de información de la aplicación (aifs).
- La quinta fase es esencial, crea los ejecutables (main) de la aplicación.
- En esta sexta y ultima fase lo que se realiza es una comprobación de que todas las anteriores han ido correctamente.

Después de haber ejecutado todos los comandos anteriores y haber visto las fases de compilación solo queda decir que el resultado de esta compilación puede encontrarse en el directorio \epoc32\release\winscw \udeb \holamundo.exe. Para hacerlo funcionar puede ejecutarse desde ahí o desde el explorador de Windows. Una vez ejecutado se abrirá el emulador y mostrara una imagen como la siguiente.

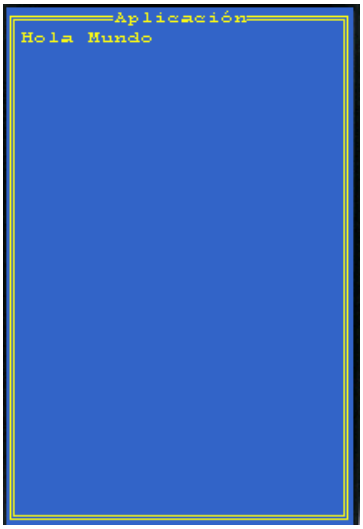


Figura 169. Captura de pantalla de la emulación de una aplicación por línea de comandos.

Compilación de una aplicación utilizando Carbide C++

Este proceso es muy sencillo, anteriormente se ha visto como crear un proyecto y también se ha visto el código de la aplicación Hola Mundo. Por lo tanto se pasará directamente a la compilación ya que se supone que se tiene Carbide C++ en funcionamiento y con el proyecto Hola Mundo creado y abierto.

En la parte superior del IDE se puede ver una barra que contiene todos los elementos que se van a utilizar, entre ellas hay un martillo, una especie de escarabajo azul y dos símbolos mas uno azul y otro verde. La siguiente imagen es de la barra en cuestión.

Figura 170. Barra de emulación y debugación de Carbide C++.

Las funcionalidades de la barra son las siguientes:

- El primer símbolo sirve para crear proyectos nuevos o añadir nuevos archivos al proyecto actual.

- El segundo y tercer símbolo que salen sin marcar sirven para guardar los cambios y para imprimir respectivamente.

- El tercer símbolo (el martillo) es el icono de compilación, en el se podrá elegir el tipo de compilación que se quiere hacer. En este caso existen 4 posibles opciones que se verán en la siguiente imagen. Estas opciones de compilación se usaran según el caso.

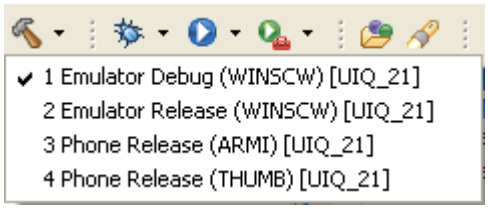


Figura 171. Captura de pantalla del desplegable del botón compilación de Carbide C++.

Para una compilación con emulador y debugador se usa la primera opción (winscw udeb), la segunda opción es para compilar con emulador pero sin poder hacer uso del debugador (modo release) y winscw urel. La tercera y cuarta opción es para compilar sin emulador, estas son utilizadas una vez que se ha comprobado que la aplicación funciona correctamente en el emulador. Sirven para generar el archivo de instalación en el teléfono. La diferencia entre ambas es que la tercera opción (ARMI) se utiliza para generar instalables para dispositivos con procesadores tanto de 16 como de 32 bits, y la cuarta opción es para procesadores de 16 bits (THUMB). Los programas THUMB son más pequeños y los ARMI son más rápidos. Lo mas eficiente para nuestro terminal es la opción de programas THUMB a si que llegado el momento elegiremos esa opción.

- El tercer símbolo es para ejecutar el debugador. Si queremos debuguear el programa lo pulsaremos y a continuación entraremos en la pantalla de debugación. Esta será vista mas adelante en profundidad.

- El cuarto símbolo es para lanzar el emulador. Una vez que se ha compilado el programa si lo pulsamos se ejecutara en el emulador.

- El quinto símbolo y último que se va a ver de momento (ya que el resto carece de utilidad importante) sirve para ejecutar las aplicaciones con herramientas externas, con el se pueden crear nuevas configuraciones para ejecutar las aplicaciones en otros modos.

Una vez vistas las funcionalidades de la barra se vera el proceso a realizar para compilar una aplicación y generar su archivo instalable en el terminal (sis).

Para realizar la compilación del proyecto habrá que pulsar el botón del martillo y seleccionar la primera opción (EMULATOR DEBUG) para poder compilar con el emulador y debugador. Una vez hecho esto simplemente habrá que irse a la pantalla superior izquierda donde se encuentran todos los archivos y pinchar con el botón derecho sobre el nombre del proyecto. En ese punto aparecerá un desplegable con todas las opciones posibles entre las cuales deberá elegirse Build Proyect.

En la siguiente imagen puede verse exactamente lo que debe aparece y cual es la opción elegida.

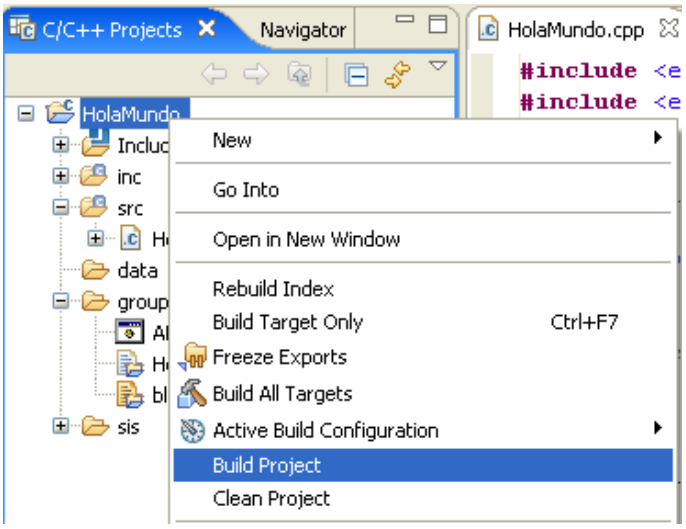


Figura 172. Captura de pantalla del desplegable para compilación de proyecto de Carbide C++.

Se deberá pulsar en la misma y Carbide C++ se encargará de compilar el proyecto. El siguiente paso es probar la aplicación en el emulador o en el caso de que hayan habido errores utilizar el debugador para subsanarlos. El uso de estas herramientas se vera mas adelante, ahora se seguirá con la compilación.

Una vez hecho esto se va a proceder a la creación del archivo de instalación en el dispositivo. Para ello se debe pinchar de nuevo en el botón del martillo y elegir la cuarta opción (Phone Release (THUMB)). De nuevo se pulsara con el botón derecho del ratón sobre el proyecto y se escogerá la opción Build Proyect para compilar en este nuevo modo.

Una vez compilado para crear el archivo .sis deberá abrirse el desplegable de la carpeta llamada sis y pinchar con el botón derecho sobre el archivo .pkg que se encuentre dentro. Ahora se deberá elegir la opción Build PKG la cual generará en esta carpeta el archivo sis.

A continuación una imagen mostrando los archivos generados después de las diversas compilaciones.

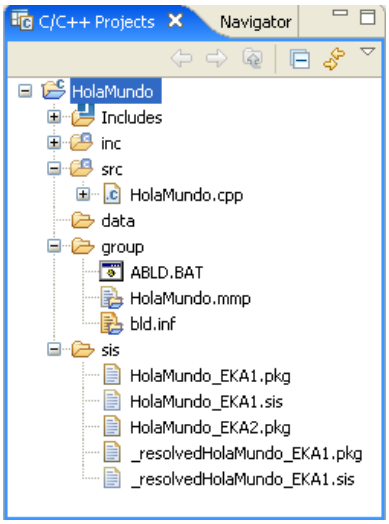


Figura 173. Captura de pantalla de la ventana de Proyectos de Carbide C++

Una vez efectuados todos los pasos podremos instalar la aplicación Hola Mundo en nuestro Sony Ericsson P800.

4.4.1.7 Emulador

El emulador es una herramienta fundamental de la plataforma Symbian OS y es vital aprender a utilizarla. Si nunca se ha utilizado el Sistema Operativo Symbian, el emulador ofrece la oportunidad de empezar a conocerlo desde la perspectiva del usuario.

Lanzando el Emulador: Una vez que se ha instalado el SDK para C++ de Symbian OS UIQ, se puede lanzar el emulador de alguna de las dos maneras siguientes:

- Desde el Explorador de Windows: hay que buscar el directorio \epoc32\release\winscw\udeb que estará dentro de la carpeta donde se haya instalado el SDK. Una vez en este punto hay que lanzar el archivo epoc.exe.
- Desde la línea de comandos: hay que abrir una línea de comandos. Para ello se debe ir al menú de inicio y en ejecutar poner cmd. Se abrirá una línea de comandos. A continuación en la línea de comandos habrá que navegar hasta el directorio adecuado en este caso poniendo \epoc32\release\winscw\udeb precedida de la letra adecuada. Una vez en este directorio habrá que escribir epoc para que se ejecute el emulador.

Existe una tercera manera de lanzar el emulador y es a partir del Carbide C++, para ello se puede hacer o bien pulsando el botón especifico de la barra del cual se pondrá una imagen a continuación o pulsando Ctrl + F11.

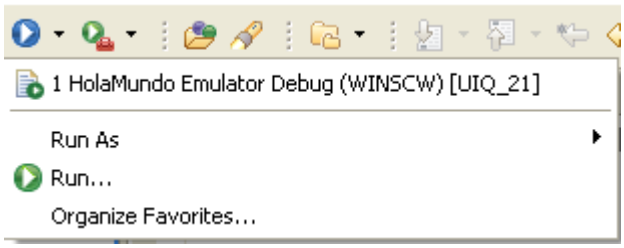


Figura 174. Captura de pantalla del desplegable con las opciones de emulación de Carbide C++.

El botón para lanzar el emulador es el azul, el primero que sale en esta imagen empezando por la izquierda.

4.4.1.9 Debugador

El debugador se utiliza para corregir errores en los programas. En este caso, Carbide C++ incorpora un debugador muy útil y con un muy buen funcionamiento. Para lanzarlo se puede utilizar su botón especifico (del cual se pondrá una imagen a continuación) o bien pulsar F11.

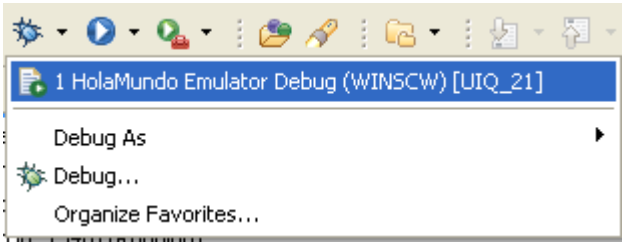


Figura 175.Captura de pantalla del desplegable con las opciones de debugación de Carbide C++.

Una vez lanzado, Carbide C++ crea una pantalla especifica que es del debugador, en ella se pueden utilizar todas sus opciones. Entre ellas se pueden destacar la ejecución de trozos concretos de código, ejecución paso a paso, introducción de breakpoints, etc.

Su uso no requiere ningún estudio específico ya que es bastante simple pero eficaz. A continuación una captura de pantalla del emulador en ejecución.

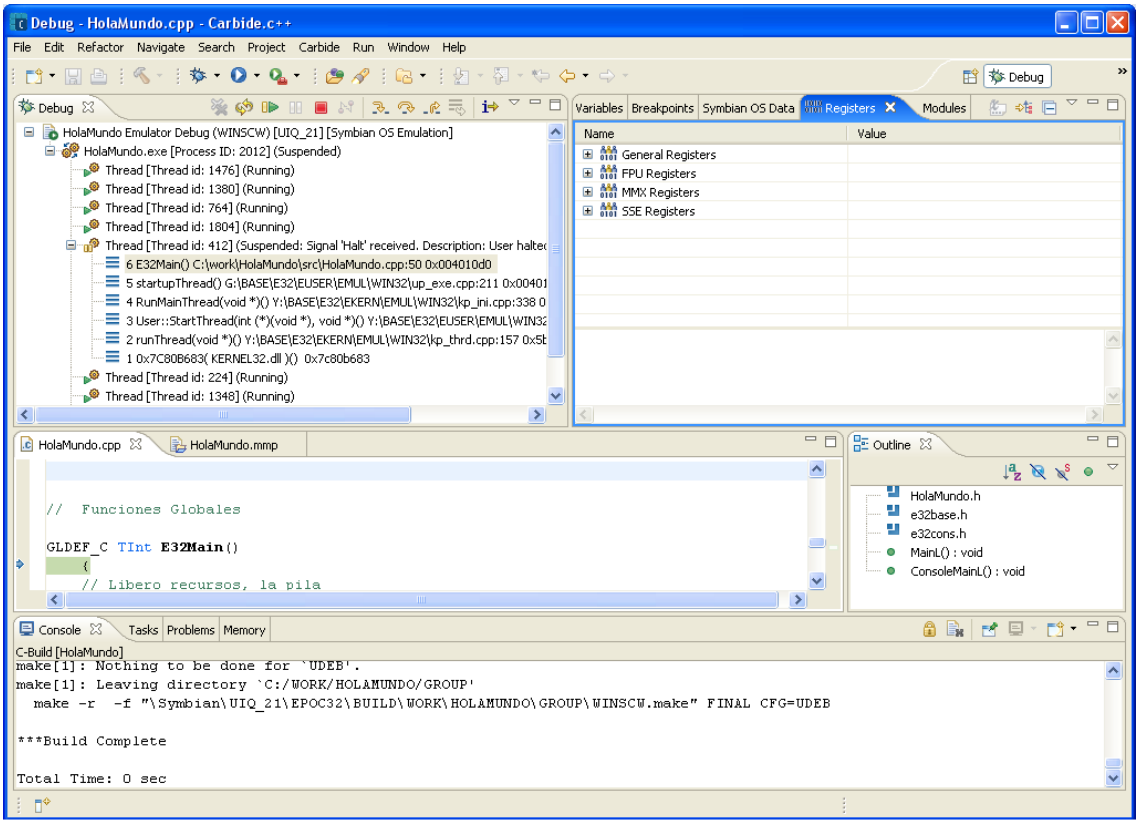


Figura 176. Captura de pantalla del entorno de debugación de Carbide C++

Como se puede ver en la imagen, la pantalla se ha dividido en 4 zonas de influencia.

En la de arriba a la izquierda se va detallando el funcionamiento del programa, como van saltando los procesos y las diferentes funciones que componen la aplicación.

Se puede observar también que ha salido una nueva barra justo debajo de la barra normal, en la cual se tiene a mano cualquier parámetro o valor que se le quiera introducir al programa como prueba en debugación.

La segunda zona esta arriba a la derecha, esta se compone de 5 pestaña con los nombres: Variables, Breakpoints, Symbian OS Data, Registers y Modules. En ellas se mostraran los parámetros que les dan nombre, las variables del programa, los breakpoints introducidos, el estado de cada registro, etc.

La tercera zona se encuentra extendida en el centro de la pantalla, esta se divide en dos subzonas, la de la parte izquierda muestra el código del programa mostrando una marca verde en el punto el cual se esta ejecutando en cada momento.

En la parte derecha se muestran las diferente funciones que componen la aplicación y va mostrando que función de la aplicación se esta ejecutando en el debugador en cada momento.

La cuarta zona es la Consola de salida, en ella se van mostrando los eventos de texto generados por el programa, al igual que los posibles errores y warnings generados por la aplicación.

4.4.2 Entrada /Salida de datos en C++

4.4.2.1 Introducción

En este capitulo se realizara un estudio de la clase RFile de Symbian con la cual podremos manejar ficheros en Symbian de una manera sencilla. Esta clase se basa en los fundamentos adquiridos de C con el uso de las funciones fopen/fread/fwrite/fclose. Se aprenderá a utilizar la clase RFile consiguiendo de este modo unos conocimientos básicos de la gestión de ficheros, su apertura, escritura y cierre.

En la siguiente figura se muestra la composición exacta de la API.

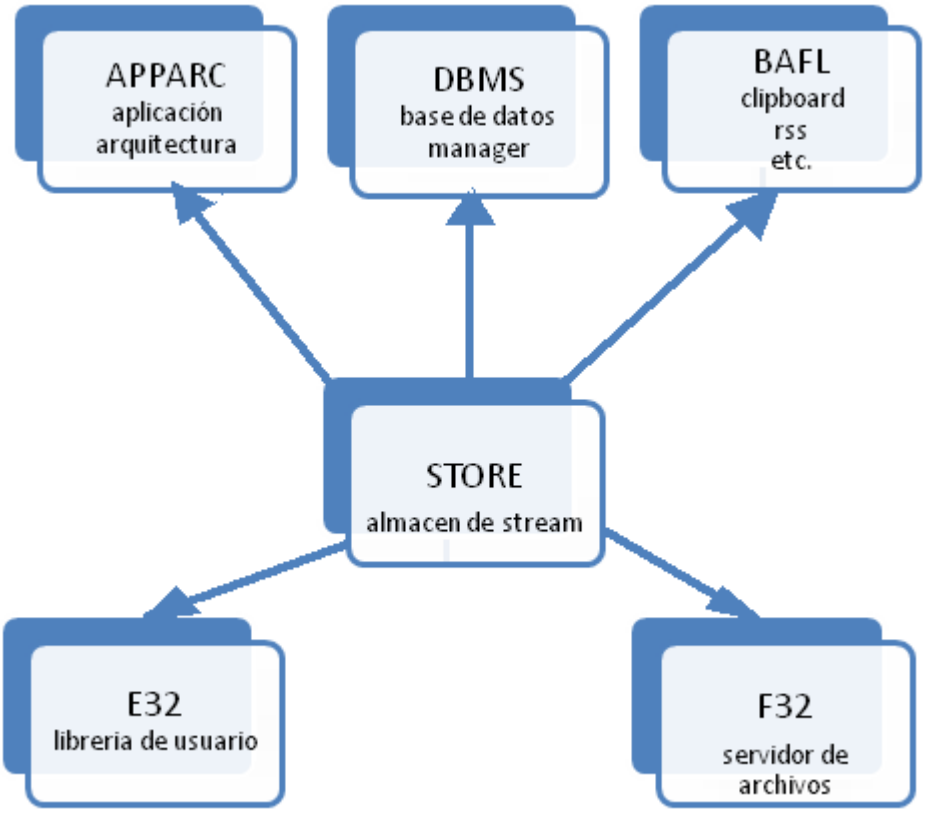


Figura 177. Diagrama con la estructura de la API de C++

4.4.2.2 RFile

Ahora se vera en profundidad esta clase, las funciones de las que se compone y cuales son sus utilidades tal y como van definidas por Symbian OS.

La descripción de esta clase como tal se puede encontrar en f32file y la librería a la que pertenece es efsrv.lib. Esta clase se encarga del manejo de cualquier tipo de archivo en Symbian OS. Su creación, apertura, escritura, cierre, etc.

La definición de un objeto RFile es la siguiente:

class RFile : public RSubSessionBase;

Las operaciones básicas que se pueden realizar con esta clase son:

- Lectura y escritura de un archivo.
- Desplazamiento hasta un punto especifico del archivo.
- Apertura y cierre de un archivo.
- Configuración de los diversos atributos y propiedades de los archivos.

Es importante recordar que para la realización de cualquier operación con archivos lo primero que hay que hacer es abrirlo. Es necesaria su apertura para poder trabajar con ellos, al igual que es obligatorio cerrar un archivo cuando se termine de utilizar ya que si no los programas generaran errores en compilación.

En la siguiente tabla se exponen las diferentes maneras de apertura y creación de un archivo.

Función	Descripción
Open()	Se utiliza para abrir un archivo en el cual queremos leer o escribir.
Create()	Sirve para crear y abrir un nuevo archivo en el cual queremos escribir. Se generará un error si el archivo ya existe.
Replace()	Con esta función se abrirá un archivo que ya existe reemplazando el contenido del mismo por otro.
Temp()	Esta función es utilizada para crear un archivo del tipo Temporal. Este archivo tendrá una duración determinada en el tiempo.

Cuando se abre un archivo se debe especificar la sesión de servicio del archivo en la cual se van a realizar las operaciones sobre el mismo. Si el archivo que esta siendo utilizado no es cerrado explícitamente en el código de la aplicación, el mismo será cerrado cuando su sesión de servicio asociada se cierre.

Lectura y Escritura.

Hay diversas variantes para la lectura y escritura de un archivo. La primera de ellas es la lectura básica (TDes && aDes), también existe la misma en modo escritura (const TDesC8& aDes). A partir de estas dos se generan el resto de los modos, dependiendo del tamaño del descriptor que les sea pasado como parámetro. Según el mismo se especificará si lo que se pretende es sobrescribir un archivo, modificar un pedazo del archivo a partir del primer byte especificado, completar un archivo asíncronamente, etc.

En la lectura y escritura de datos que se quieren enviar al archivo los descriptores de los métodos utilizados para ello serán tipos binarios (TDes8, TDesC8).

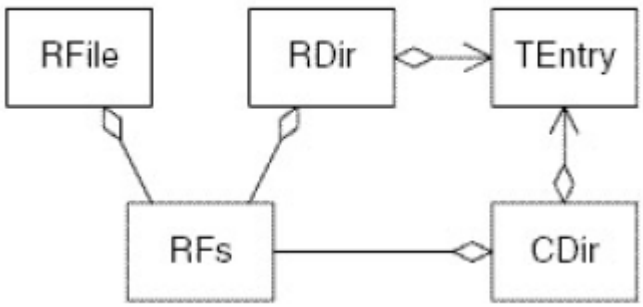


Figura 178. Diagrama con la estructura de RFile.

Las funciones definidas en RFile son muy variadas, por ello se van a organizar en los siguientes puntos las más importantes según su función.

Apertura y cierre de un archivo.

- *Open():* *IMPORT_C TInt Open(RFs &aFs, const TDesC &aName, TUint aFileMode);*

  Abre un archivo para su lectura o escritura.

- *Close():* *IMPORT_C void Close();*

  Cierra un archivo.

- *Create():* *IMPORT_C TInt Create(RFs &aFs, const TDesC &aName, TUint aFileMode);*

  Crea un archivo nuevo y lo abre para su escritura. Si el nombre del archivo a crear coincide con un archivo ya creado devuelve un error.

- *Temp():* *IMPORT_C TInt Temp(RFs &aFs, const TDesC &aPath, TFileName &aName, TUint aFileMode);*

  Crea y abre un archivo temporal nuevo para su posterior lectura o escritura.

Lectura/Escritura de un archivo.

- *Read():* Hay diversas formas de lectura de un archivo, a continuación se exponen todas las posibilidades.

  a. *IMPORT_C TInt Read(TDes8 &aDes) const;*
  IMPORT_C void Read(TDes8 &aDes, TRequestStatus &aStatus) const;

  Se utiliza para leer desde la posición actual de un archivo. El primero lo realiza en modo Síncrono y el segundo en modo Asíncrono.

  b. *IMPORT_C TInt Read(TDes8 &aDes, TInt aLength) const;*
  IMPORT_C void Read(TDes8 &aDes, TInt aLength, TRequestStatus &aStatus)

  Lee el número de bits determinado por la variable aLength. El primero en modo Síncrono y el segundo en modo Asíncrono.

  c. *IMPORT_C TInt Read(TInt aPos, TDes8 &aDes) const;*
  IMPORT_C void Read(TInt aPos, TDes8 &aDes, TRequestStatus &aStatus) const;

  Lee de un archivo específico aplicándole un offset. El primero en modo Síncrono y el segundo Asíncrono.

  d. *IMPORT_C TInt Read(TInt aPos, TDes8 &aDes, TInt aLength) const;*
  IMPORT_C void Read(TInt aPos, TDes8 &aDes, TInt aLength, TRequestStatus &aStatus) const;

Lee un número de bits especificado y aplicándole un offset. El primero en modo Síncrono y el segundo en modo Asíncrono.

- *ReadCancel():*

  a. *IMPORT_C void ReadCancel(TRequestStatus &aStatus) const;*

  Con esta función se pude cancelar una petición de lectura específica.

  b. *IMPORT_C void ReadCancel() const;*

  Cancela todas las peticiones de lectura.

- *Write():*

  a. *IMPORT_C TInt Write(const TDesC8 &aDes);*
  IMPORT_C void Write(const TDesC8 &aDes, TRequestStatus &aStatus);

  Escribe en el archivo, la primera es para el modo Síncrono y la segunda para el modo Asíncrono.

  b. *IMPORT_C TInt Write(const TDesC8 &aDes, TInt aLength);*
  IMPORT_C void Write(const TDesC8 &aDes, TInt aLength, TRequestStatus &aStatus);

  Escribe una porción del descriptor del archivo pasándole un offset. El primero es para el modo Síncrono y el segundo para el modo Asíncrono.

  c. *IMPORT_C TInt Write(TInt aPos, const TDesC8 &aDes);*
  IMPORT_C void Write(TInt aPos, const TDesC8 &aDes, TRequestStatus &aStatus);

  Escribe en el archivo con un offset específico. El primero Síncrono y el segundo Asíncrono.

  d. *IMPORT_C TInt Write(TInt aPos, const TDesC8 &aDes, TInt aLength);*
  IMPORT_C void Write(TInt aPos, const TDesC8 &aDes, TInt aLength, TRequestStatus &aStatus);

  Escribe un número determinado de bytes en el archivo. Primero modo Síncrono y el segundo modo Asíncrono.

Ahora por último se creará una tabla en la que se expondrán el resto de funciones de la clase. Entre ellos están las funciones accesoras a los archivos, funciones de actualización y cambios de modo.

Función	Descripción
TInt Adopt(RFs &aFs, TInt aHandle)	Toma el control sobre un archivo ya abierto.
TInt AdoptFromClient(const RMessage2 &aMsg, TInt aFsHandleIndex, TInt aFileHandleIndex)	Permite que un servidor adopte un archivo ya abierto de un cliente.
TInt AdoptFromServer(TInt aFsHandle, TInt aFileHandle)	Permite que un cliente adopte un archivo ya abierto de un servidor.
TInt AdoptFromCreator(TInt aFsIndex, TInt aFileHandleIndex)	Permite que un servidor adopte un archivo ya abierto por un proceso de un cliente.
TInt Att(TUint &aAttValue) const	Devuelve los atributos del archivo.
TInt ChangeMode(TFileMode aNewMode)	Esta función permite impedir el acceso de sólo lectura, sin tener que cerrar y volver a abrir el archivo.
TInt Drive(TInt &aDriveNumber, TDriveInfo &aDriveInfo) const	Devuelve información de la unidad donde se encuentra ubicado el archivo.
TInt Duplicate(const RFile &aFile, TOwnerType aType=EOwnerProcess)	Realiza un duplicado del archivo pasado como atributo en otro hilo de ejecución.
TInt Flush()	Fuerza la ejecución de los métodos anteriores y vacia los bufferes.
TInt FullName(TDes &aName) const	Devuelve el nombre completo del archivo.
TInt Lock(TInt aPos, TInt aLength) const	Bloquea una parte del archivo al uso público.
TInt Modified(TTime &aTime) const	Devuelve la fecha y hora local en que fue modificado por última vez el archivo.
TInt Name(TDes &aName) const	Devuelve la parte final del nombre del archivo.
TInt Rename(const TDesC &aNewName)	Renombra un archivo.
TInt Replace(RFs &aFs, const TDesC &aName, TUint aFileMode)	Abre el archivo especificado y reemplaza su contenido. Si el archivo no existe lo crea y escribe.
TInt Seek(TSeek aMode, TInt &aPos) const	Fija la posición actual del archivo.
TInt Set(const TTime &aTime, TUint aSetAttMask, TUint aClearAttMask)	Establece los atributos del archivo, asi como la hora y la fecha de modificación.
TInt SetAtt(TUint aSetAttMask, TUint aClearAttMask)	Establece o limpia los atributos de un archivo utilizando enmascaramiento.
TInt SetModified(const TTime &aTime)	Fija la hora y la fecha del último archivo que fue modificado.
TInt SetSize(TInt aSize)	Establece el tamaño del archivo.
TInt Size(TInt &aSize) const	Devuelve el tamaño del archivo actual.
TInt TransferToClient(const RMessage2 &aMsg, TInt aFileHandleIndex) const	Transfiere un archivo que actualmente esta en un servidor a un cliente.
TInt TransferToProcess(RProcess &aProcess, TInt aFsHandleIndex, TInt aFileHandleIndex) const	Trensfiere un archivo abierto actualmente a otro proceso.
TInt TransferToServer(TIpcArgs &aIpcArgs, TInt aFsHandleIndex, TInt aFileHandleIndex) const	Transfiere un archivo que actualmente esta en un cliente a un servidor
TInt UnLock(TInt aPos, TInt aLength) const	Desbloquea una parte bloqueada del archivo.

4.4.2.3 Implementación de ejemplo con RFile

En este punto se expone un pequeño ejemplo de la utilización de la clase vista y sus funciones. Se van a poner los códigos de dos clases, por un lado la clase cabecera (File.h) en la cual se definirán las funciones que se van a utilizar, por otro lado el código fuente con la implementación de las diversas funciones. Estas dos clases pueden ser añadidas a cualquier proyecto en el cual se necesite el manejo de ficheros. También habrá unas imágenes con la emulación de un proyecto en la cual se utilizan estas clases.

Clase File.h

```
#ifndef FILE_H
#define FILE_H

#include <f32file.h>

class File {
 public:
       enum OpenMode {OMRead = 1,OMWrite = 2,OMText = 4,OMCreate =
       8,OMReplace = 16,OMOpen = 32};

       bool Open(const TDesC &,unsigned int mode);
       void Close();
       int Read(void *buff,int length);
       int Write(void *buff,int length);
       void Seek(TSeek mode,int offSet);

private:
       RFs fsSession;
       RFile rFile;
};

#endif
```

Clase File.cpp

```
#include "File.h"

bool File::Open(const TDesC &name,unsigned int mode){
 TInt mask = 0;
 TInt err;

 fsSession.Connect();

if(mode&OMText) mask = EFileStreamText;
if(mode&OMRead) mask |= EFileRead;
if(mode&OMWrite) mask |= EFileWrite;

if(mode&OMCreate) err =  rFile.Create(fsSession,name,mask);
else if(mode&OMReplace) err =  rFile.Replace(fsSession,name,mask);
else if(mode&OMOpen) err = rFile.Open(fsSession,name,mask);

 if(err != KErrNone){
       fsSession.Close();
       return false;
 }
```

```
 else
       return true;
}

void File::Close(){
 rFile.Flush();
 rFile.Close();
 fsSession.Close();
}

int File::Read(void *buff,int length){
 TPtr8 ptr((unsigned char*)buff,length);
 rFile.Read(ptr,length);
 return ptr.Length();
}

int File::Write(void *buff,int length){
 TPtr8 ptr((unsigned char*)buff,length,length);
 rFile.Write(ptr);
 return ptr.Length();
}

void File::Seek(TSeek mode,int offSet){
 rFile.Seek(mode,offSet);}
```

Figura 179. Captura de pantalla de la emulación de la aplicación File con Carbide C++.

4.4.3 Principales APIs de C++ para Symbian OS

4.4.3.1 Introducción

En este apartado se va a hacer un repaso sobre las principales APIs de C++ para Symbian OS. En el se verán las diferentes clases con sus respectivas funciones para el dibujo y manejo de gráficos en C++, además de los diferentes eventos que se pueden generar, captura de los mismos, comandos utilizables para ello, etc. En definitiva se van a repasar de manera concisa las librerías básicas que se pueden necesitar a la hora de realizar una aplicación en C++ sobre Symbian OS.

Como ejemplo se realizará al final una aplicación, exactamente se realizará un Hola Mundo pero con una GUI avanzada, haciendo que este Hola Mundo tenga múltiples modos de visualización, uso de una animación, uso de multihilo en pantalla, tratamiento de eventos de teclado y pantalla (porque esta hecho sobre UIQ), etc. Usando casi todo lo que se va a ver en este capitulo.

4.4.3.2 Manejo de Gráficos en pantalla.

Se va a comenzar viendo los siguientes apartados:

- *Dibujo de gráficos*: como manejar los gráficos en la pantalla, trabajando con un ejemplo de código básico.
- *La API CGraphicsContext*: se realizará un estudio de esta clase, ya que contiene las funciones básicas de dibujo.
- *El paradigma modelo-vista-controlador (MVC):* servirá para introducir la relación entre los gráficos y su interacción con Symbian OS.
- *División de la pantalla:* en este apartado se verá como poder dividir la pantalla y controlar cada una de las subpantallas generadas.
- *Efectos especiales:* Como su propio nombre indica, aquí se incluirán aquellas funciones destinadas a generar tipos de efectos poco corrientes o especiales.

Dibujo de gráficos: El GUI de una aplicación puede presentar diversas apariencias según las funciones que se usen para implementarlo, por ello antes de comenzar a realizar una interfaz gráfica para una aplicación es necesario hacerse una serie de preguntas básicas tales como:

- ¿Qué fuente quiero usar?
- ¿Qué colores utilizaré para el fondo de pantalla y para los cuadros de texto?
- ¿Necesitaré poner bordes a la pantalla o mejor utilizo frames?
- ¿Qué tamaño de pantalla voy a usar para que la visualización de la aplicación sea la adecuada?

Es importante tener claras todas estas decisiones antes de comenzar con la implementación, ya que el aspecto de la aplicación dependerá totalmente de ello.

Controles: Desde la perspectiva de un programador de aplicaciones para Symbian OS, todos los dibujos de la pantalla necesitan tener un control. El control es un área rectangular que ocupa toda la pantalla. La clase base para todos los controles es CCoeControl, la cual esta definida en un componente llamado CONE perteneciente a Symbian OS. En la imagen de abajo se puede ver un ejemplo de pantalla en UIQ con SDK 3.1.

Figura 180. Captura de pantalla del entorno UIQ usando el emulador de Carbide C++.

Esta pantalla se compone de tres ventanas principales, la parte de arriba sirve para mostrar iconos tales como la batería o la potencia de la señal que se recibe, la de en medio es la pantalla en si y la de abajo es donde se agregan los botones de manejo de las aplicaciones que se muestran en la pantalla de en medio. Como se puede ver en esta pantalla, existen dos posibles interacciones con la misma en dos de sus pantallas:

- En la pantalla de en medio el controlador se encarga de recoger los eventos generados al elegir una aplicación concreta.
- En la pantalla de abajo, o de botones, el controlador se encarga de realizar las acciones posibles con cada aplicación que se pueda seleccionar y mostrar las opciones de cada una.

Función Draw(): En Symbian OS todos los dibujos están incluidos dentro del contexto gráfico (GC). Ahora se introducirán los modos de uso del mismo.

Dentro del contexto gráfico: La función principal dentro de este contexto gráfico (GC) es SystemGc() que se encuentra en CCoeControl. Un ejemplo de su uso es:

```
CwindowGc& gc = SystemGc();
```

Todas las clases del contexto gráfico derivan de CGraphicsContext. Estas clases derivadas son usadas para dibujar en una fuente gráfica determinada y todas sus implementaciones están especificadas en la clase base. Se puede limpiar la pantalla usando el contexto gráfico y la función:

gc.Clear();

El contexto gráfico es una noción común dentro del mundo de las computadoras. Windows usa "device context"; Java usa "Graphics Objects". Como se puede apreciar esta noción esta extendida en todos los sistemas.

Dibujo de un rectángulo: Las siguientes tres líneas de código que se van a introducir sirven para dibujar un borde rectangular al área "vista" de la aplicación en la pantalla:

TRect rect = Rect();

CCoeControl::Rect() proporciona las coordenadas que va a ocupar el rectángulo al controlador del mismo. Estas coordenadas son dadas en función de la ventana que es controlada por el controlador. Las coordenadas usan CWindowGc para dibujar las funciones.

rect.Shrink(10, 10);

Este uso de Shrink hace que el rectángulo que se dibuja sea 10 pixels más pequeño que la pantalla total por todos los márgenes, por arriba, abajo, izquierda y derecha. TRect contiene muchas funciones útiles e interesantes como las que se irán viendo después.

gc.DrawRect(rect);

Esta función dibuja un rectángulo usando las propiedades básicas (default) del contexto gráfico. Estas propiedades son:

- El bolígrafo pinta en color negro, el tamaño es de un pixel, las líneas son solidas.
- La brocha esta puesta a null, por tanto el rectángulo no tiene color interior.

Estas propiedades pueden ser cambiadas en cualquier momento haciendo uso de los métodos accesores que proporciona la clase Draw().

Dibujo de texto: Ahora se va a ver como se puede dibujar texto en la pantalla centrado en el rectángulo. Para comenzar se va a centrar el rectángulo a 1 pixel de los márgenes para tener mas espacio de escritura.

rect.Shrink(1, 1);

Ahora para elegir una fuente se utiliza Uikon el cual es el encargado de definir las diversas fuentes:

cons CFont font = iEikonEnv->TitleFont();*

Esta es la primera vez que se ve la clase CFont, solo decir de ella que se puede utilizar para elegir cualquier tipo de fuente, el tamaño, el estilo (negrita, cursiva, itálica), atributos, etc. En este caso se ha usado TitleFont proveniente de Uikon, esta es la que se suele utilizar para la barra de títulos de las cajas de diálogos y sus atributos principales son que es negrita y grande.

Para poder manejar este texto en la pantalla es necesario un puntero que apunte a font del tipo:

gc.UseFont(font);

Ahora vamos a ver como se puede dibujar el texto centrado en el rectángulo:

```
TInt baseline = rect.Height() / 2 + font->AscentInPixels() / 2;
gc.DrawText(*iHolaMundo, rect, baseline, CGraphicsContext::ECenter);
```

La función DrawText() se encargará ahora de dibujar el texto seleccionado en el rectángulo utilizando el lápiz del GC, la fuente seleccionada y el área elegida para ello. El último parámetro especifica la justificación del texto en el rectángulo, CGraphicsContext::ECenter indica que el texto tendrá una justificación horizontal y centrada.

Justificación Vertical: Este tipo de justificación es más complicado de calcular ya que la función DrawText() no calcula la línea de base, y en el caso de querer aplicar este tipo de justificación habría que calcularla por uno mismo. Afortunadamente, el algoritmo para calcularla es bastante sencillo y no depende del texto. Hay que especificar el tamaño en pixels de la línea de base desde la parte superior a la parte inferior del rectángulo que enmarca el texto y después añadir el posible desplazamiento desde la línea de base de la fuente, tanto superior como inferior. En la siguiente imagen se pueden ver las diferentes medidas que se le puede suministrar al algoritmo para la creación de Justificación Vertical.

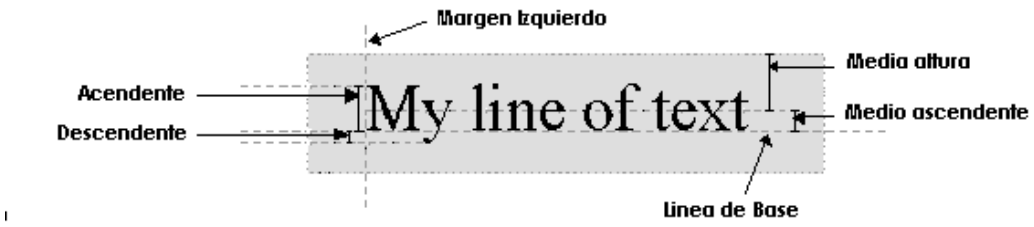


Figura 181. Esquema de las coordenadas necesarias para dibujar texto con un justificado específico.

Una vez escrito el texto necesario, es importante descartar la fuente utilizada, para liberar el uso de recursos de la misma y su posterior utilización por otro proceso. Esto se hace añadiendo la siguiente línea al código:

gc.DiscardFont();

El API CGraphicsContext: Todas las clases gráficas en C++ para Symbian heredan de la clase CGraphicsContext. Para describir en profundidad esta API se incluye el siguiente diagrama UML.

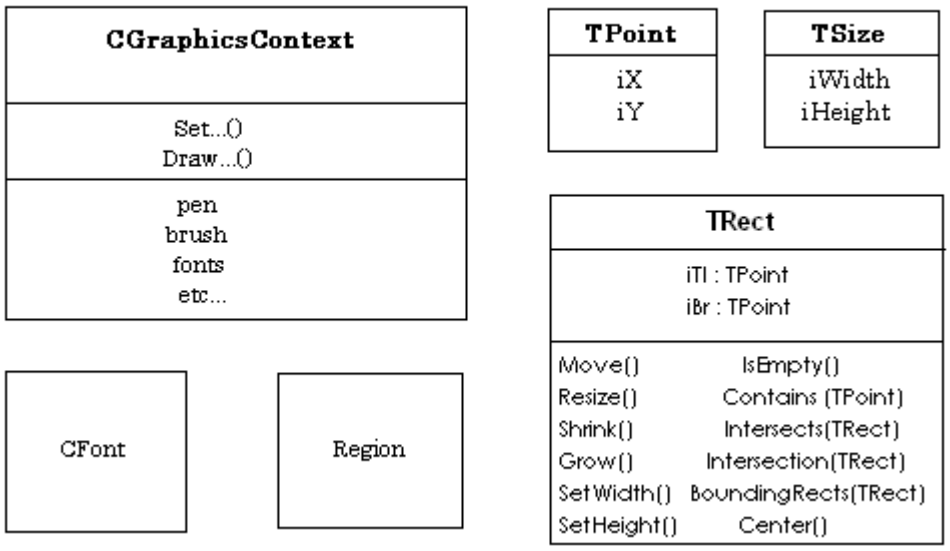


Figura 182. Diagrama de la clase CGraphicsContext de C++ en Symbian OS.

CGraphicsContext contiene las funciones básicas de dibujo y esta definido en gdi.h. Todos los dibujos son realizados utilizando alguna configuración del lápiz, el pincel o de las fuentes y una vez hecho esto son adheridos a la pantalla.

Coordenadas en las Clases: Los gráficos son dibujados en las ventanas utilizando un sistema de coordenadas definido en pixels. Cada pixel viene marcado por un punto (x, y) perteneciente a la pantalla. El punto (0, 0) esta ubicado en el margen superior izquierdo, y a partir de este punto si se incrementa la x el punto se desplazará hacia la derecha y si se incremente la y el punto se desplazará hacia abajo.

Las clases para realizar gráficos en 2D, como los puntos, rectángulos y regiones están definidas en e32std.h:

- TPoint contiene las coordenadas iX e iY.

- TRect contiene dos puntos. iT1 para la esquina superior izquierda e iBr para la esquina inferior derecha.
- TSize contiene las dimensiones iWidth e iHeight.

Todas estas clases poseen una gran cantidad de métodos constructores, operadores y funciones para manipular y combinar los diferentes tipos de gráficos. Pero no se pueden usar las funciones accesoras set/get para acceder a las coordenadas de un punto por ejemplo ya que las funciones anteriores son incapaces de encapsular sus miembros.

Los dos puntos que están definidos en TRect pueden ser interpretados por una implementación específica en cada caso. Una interpretación común de la misma suele ser la de situar el punto IT1 en la esquina superior izquierda pero dentro del área de dibujo y el punto iBr en la esquina inferior derecha pero fuera del rectángulo de dibujo. Podemos ver exactamente esto en la figura que viene a continuación.

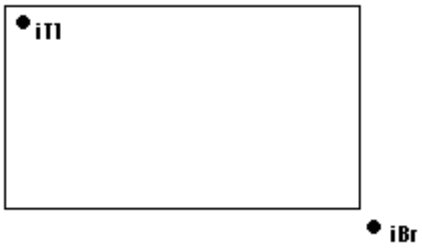


Figura 183. Esquema para entender el funcionamiento de las coordenadas en C++.

Configurando el Contexto Gráfico: CGraphicsContext contiene varios ítems importantes definidos que se pueden usar para dibujar funciones:

1. *Lápiz (Pen):* El lápiz define el modelo de dibujo (color y estilo). Este es usado para dibujar líneas, márgenes y texto. Para controlar el lápiz se pueden usar básicamente tres funciones, SetPenColor(), SetPenStyle() y SetPenSize. Por definición a no ser que sean cambiadas estas características el lápiz esta configurado en negro, solido y a un pixel cada tick (esto quiere decir que pinta un pixel cada vez que se pica con el ratón).

2. *Pincel (Brush):* El pincel define el aspecto y fondo del patrón de colores utilizado. El pincel puede estar puesto a null, solido, patrón en forma de trama o bitmap.

   Para controlar esta herramienta (pincel) se pueden utilizar como en el caso del lápiz varias funciones para controlarlo. Estas funciones básicamente son SetBrushStyle(), SetBrushColor, SetBrushOrigin(), SetBrushPattern() y DiscardBrushPattern().

   Por definición la configuración inicial del pincel es null y origen cero.

3. *Fuente (Font):* La fuente como su propio nombre indica define la fuente usada para pintar el texto. Esta se puede especificar pasándole una *CFont a CGraphicsContext.

   Se puede destacar que el CONE posee una fuente (iCoeEnv -> NormalFont()) mientras que Uikon posee varias (iEikonEnv -> TitleFont(), LegendFont(), SymbolFont(), AnnotationFont() y DenseFont()). Como los dos ítems anteriores el ítem fuente posee varios métodos para manejarlo entre los que destacan UseFont() el cual sirve para definir una fuente a utilizar, DiscardFont() que sirve para dejar de utilizar la fuente que esta actualmente seleccionada, SetUnderlineStyle() y SetStrikethroughStyle() que se utilizan para definir el algoritmo de enlace de la fuente en uso.

   Por definición la configuración inicial de Font es sin fuente designada y sin algoritmo designado, es decir, todo a null.

4. *Posición Actual (Current position):* Este ítem es definido por la función MoveTo() y varios funciones para desplazar la x DrawXxxTo() y por MoveBy() y sus correspondientes DrawXxxBy(). Estas funciones afectan a la función DrawPolyLine().

   Su configuración inicial es el punto (0, 0).

5. *Origen (Origin):* Origen define el punto de origen desde el cual se quiere comenzar a pintar dentro de la pantalla. Contiene una función principal SetOrigin() para controlarlo.

   Su configuración inicial también es el punto (0, 0).

6. *Región de enlace (Clipping Region):* Sirve para definir la región en la cual se quiere adherir un gráfico. Se puede especificar un rectángulo o un área mas específica utilizando las funciones SetClippingRect() o CancelClippingRect(). Por definición su configuración inicial esta puesta a null, es decir, sin áreas definidas.

7. *Justificación (Justification):* Antes ya se vieron los tipos de justificaciones y las maneras de tratarlas, estas justificaciones se manejan con la función DrawText() y se utiliza el objeto FORM de Symbian OS para crear los textos. Para establecer la

configuración inicial de todos los parámetros anteriormente vistos incluido este último se utiliza la función Reset().

El paradigma modelo-vista-controlador (MVC): En un programa que contiene una interfaz GUI, todos los dibujos son controlados utilizando controladores. Se puede definir un controlador por cada ítem insertado en la pantalla, consiguiendo así una multiárea en la cual todo es manejable y configurable. En un modelo de este tipo hay que tener en cuenta los siguientes puntos:

- La clase de control de Draw() es llamada cuando es necesario pintar algo en pantalla.
- Draw() otorga un contexto gráfico usando SystemGc().
- Para definir un área de dibujo se necesita utilizar la función Rect().

El problema es que no es tan sencillo como aquí se esta exponiendo, ya que es necesario el uso de redraw además de draw, este sirve para redibujar las áreas en las cuales se producen cambios o para redibujar cuando el sistema lo requiere. El sistema de redibujo entra en funcionamiento cuando:

- La ventana es construida por primera vez.
- La ventana, o parte de ella, es oscurecida por otra aplicación o por un cuadro de dialogo que es puesto encima de ella.

La aplicación inicializa redraw cuando ocurre una de las siguientes acciones:

- La aplicación cambia el contenido de la pantalla y quiere mostrar esos cambios actualizando la pantalla.
- La aplicación cambia los parámetros de dibujo (color, barras de scroll, o zoom) y quiere mostrar esos cambios actualizando la pantalla.

Para entender completamente el uso de redraw es imprescindible comprender el paradigma modelo-vista-controlador, el cual se verá a continuación.

Este paradigma es un estandar para el uso de gráficos en todos los lenguajes de programación. Dibujar funciones simplemente consiste en dibujar los modelos de datos; ellos no aportan cambios al mismo. Si se pretende cambiar algo, se deben usar otras funciones y entonces una vez cambiadas las funciones adecuadas llamar a las funciones de Draw para que reflejen los cambios en la pantalla. Esto es básicamente el paradigma MVC:

- El modelo son los datos que manipula el programa, es independiente de la entidad; si el programa es del tipo cliente-servidor, el modelo usualmente se corresponde con los datos persistentes del programa.

- La vista es a través de lo cual los usuarios pueden contemplar el modelo. La vista usa el modelo para dibujarse a si misma.

- El controlador es la parte del programa que actualiza el modelo y que ordena a la vista que se redibuje cuando se producen cambios en la misma, es decir, el controlador coordina los cambios en el modelo y la vista.

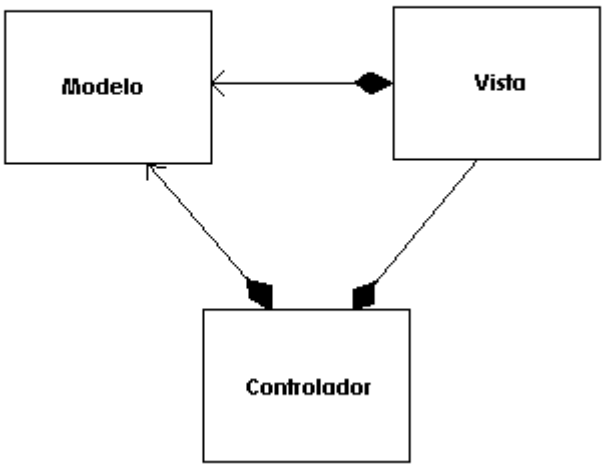


Figura 184. Esquema del paradigma Modelo-Vista-Controlador.

Dependiendo del programa de diseño las relaciones del MVC serán más o menos fuertes. Hay programas de diseño que obligan mas a la vista a actualizarse, cada mas mínimo cambio y otros que son mas débiles o menos susceptibles a cambios y permiten que tarde mas la actualización de la vista.

El paradigma MVC dicta los usos de las interrelaciones entre el modelo, vista y controlador, pero no fuerza la relación específicamente.

División de la pantalla: En este apartado se tratarán con exactitud los controladores de la pantalla, con los cuales se pueden definir varias áreas dentro de la misma y tratar cada una por separado. Manejando sus posibles eventos generados.

Symbian OS es un sistema en el cual varias aplicaciones pueden estar corriendo concurrentemente sobre el mismo. La pantalla puede ser dividida para que cada una de estas aplicaciones funcione en una parte de la misma. Symbian implementa esto utilizando el servidor de ventanas.

Las aplicaciones pueden dibujar en una o en varias pantallas a la vez. El servidor de ventanas controla la interacción de las aplicaciones con las ventanas y se encarga de mostrar u ocultar en cada momento la adecuada.

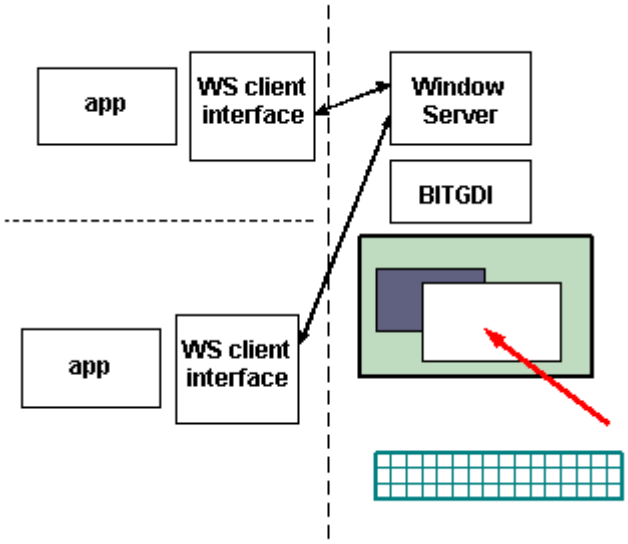


Figura 185. Esquema de la interrelación del servidor de ventanas con las aplicaciones.

Las aplicaciones no siempre hacen un uso correcto de la pantalla y de los componentes de la misma. Estos componentes están incluidos en la vista de la aplicación, en la barra de botones y el resto de los ornamentos que componen una pantalla (ventanas de dialogo, menús, etc.).

Las aplicaciones usan el control para estos componentes. Algunos controles (como las ventanas de dialogo) usan la pantalla completamente pero otros simplemente residen en una parte de la misma o en ventanas ocultas.

CONE: Todos los clientes de un GUI usan CONE, esta es una herramienta de control que provee al framework de los elementos básicos para controlar y para realizar comunicaciones con el servidor de ventanas.

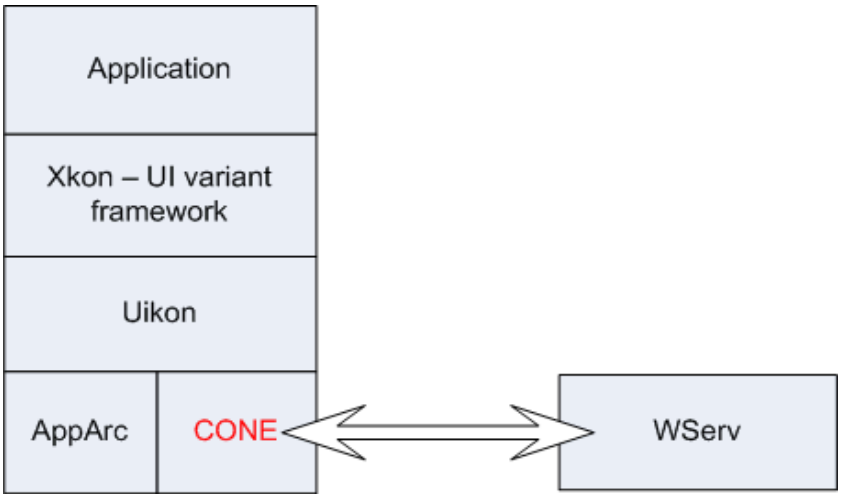


Figura 186. Diagrama de las capas de una aplicación y donde se encuentra CONE en las mismas.

El servidor de ventanas mantiene las ventanas usadas por la aplicación. Se encarga de ajustar sus coordenadas y sus posiciones en la pantalla. También se encarga de redibujar todo el entorno cuando una de las ventas es redimensionada o movida a otra parte de la pantalla.

Es importante resaltar que para cada una de las ventanas el servidor de ventanas guarda una región no valida, en la cual no se podrá situar ningún objeto u elemento. Cuando se intenta alojar un objeto en esta región de la pantalla el servidor de ventanas crea un evento de redibujo el cual es enviado al manejador de eventos de las ventanas para que la aplicación pueda redibujar la ventana correctamente.

Todas las aplicaciones funcionan como cliente con respecto al manejador de ventanas, no es importante explicar con una gran cantidad de detalles la aplicación cliente-servidor para GUIs ya que la interfaz del cliente viene encapsulada dentro del CONE.

CONE asocia uno o varios controles con cada ventana y con el manejador de eventos del servidor de ventanas. Para redibujar una ventana y mantener todos sus controles y eventos actualizados siempre se llama a la función Draw(), que tiene esa función específica.

Efectos especiales: El servidor de ventanas provee el uso de efectos especiales para las aplicaciones. Estos efectos incluyen:

- *Sombras:* Pueden ser usadas en muchas circunstancias (detrás de los diálogos, en los menús, en las listas, etc.). Para usar una sombra hay que especificar en que ventana quiere usarse la sombra y que tamaño tiene la misma. Normalmente las sombras se usan detrás de las ventanas. Para implementar una sombra el servidor de ventanas usa BITGDI para definir la región donde se pondrá la sombra. Lo que realiza BITGDI es oscurecer los colores de la ventana seleccionada para aplicar la sombra. Para el uso de las misma se utiliza la función AddWindowShadow() que se encuentra en CEikonEnv.

- *Animaciones:* A veces se quiere realizar un dibujo en movimiento, para ello Symbian OS provee ciertos métodos que se ejecutan en relación con un Timer, el cual se encargará de manejar la velocidad y el tiempo en el que esta aplicación cambiara imágenes o las desplazará por la pantalla. Las animaciones no están contenidas dentro del paradigma MVC por ello hay que utilizar un soporte especial para ellas.

  El punto clave para el uso de animaciones reside en la secuencia de flujo – parada. CONE provee una función especifica para el uso de esta secuencia, CCoeEnv::Flush(), la cual toma un tiempo de intervalo especificado en microsegundos. La siguiente función es un ejemplo del uso de la secuencia flujo – parada:

  iCoeEnv -> Flush(200000):

Muchas de las aplicaciones hacen uso de objetos activos los cuales se integran en la aplicación, estos objetos activos generan eventos que son recogidos y tratados por el servidor de ventanas.
Estas aplicaciones con animaciones son una parte del servidor de ventanas, en concreto se encuentran en animation DLL. Ellos requieren una gran atención del servidor de ventanas para poder ejecutarse y objetos activos de corta duración ya que los de larga duración pueden generar excepciones irrecuperables.

- *Uso de las llaves para debug:* Estas llaves para debug se encuentran en Uikon y sirven para manejar los usos de memoria y controlar las aplicaciones con un conjunto concreto de teclas. En la siguiente tabla se describen las diversas combinaciones existentes y su función.

Teclas	**Efecto**
Ctrl + Alt + Shift + M	Crea una ventana móvil que se puede mover por toda la pantalla, forzando a redibujar cada parte de la misma.
Ctrl + Alt + Shift + R	Provoca que la aplicación completa se redibuje.
Ctrl + Alt + Shift + F	Provoca que el servidor de ventanas realice un auto-flush. Con ello se puede comprobar que tal realiza el paradigma de flujo – parada la aplicación. Observando si hay efectos extraños, como por ejemplo animaciones que tapan a otras partes necesarias de la aplicación, desarrollo de las animaciones, efectos concretos de las sombras, etc.
Ctrl + Alt + Shift + G	Esta conjunción de teclas sirve para desactivar el auto-flush.

- *Scrolls*: El servidor de ventanas soporta el scrolling, esto significa que si la ventana que se quiere mostrar es mas grande que la pantalla se generarán una barras tanto en el lateral como en la parte inferior para poder mover la pantalla a una parte especifica de la ventana y así poder visualizar toda la ventana. Para manejar los scrolls se puede utilizar la función DrawXxxNow().

  Las barras de scroll pueden ubicarse donde se quiera, tanto arriba, abajo izquierda y derecha de la pantalla.

Es importante resaltar que todos estos efectos dependen de la implementación Symbian que se este usando. Por ejemplo UIQ no soporta el uso de sombras en sus aplicaciones.

4.4.3.3 Interacción con los gráficos

En el apartado anterior se vio como Symbian OS usa el MVC para dibujar en pantalla lo que necesita, en este apartado se verá como interactuar con las aplicaciones usando el control.

En teoría no es muy complicado. Cada control provee una función Draw() virtual para dibujar y esta misma ofrece dos funciones virtuales para manejar las interacciones con las aplicaciones, HandlePointerEventL() para manejar eventos de punteros y OfferKeyEventL() para manejar eventos de teclado. Simplemente se debe aprender a combinar estas dos funciones para poder manejar todos los posibles eventos que se generen. Ahora se introducirá en varios puntos unas pequeñas introducciones de los eventos que se pueden generar:

- Eventos de teclado: Normalmente están asociados a los cursores y más específicamente vienen definidas por el foco del escuchador. Este se refiere al punto donde se asocia un generador de eventos con el captador de dichos eventos.
- Eventos de puntero: Suelen ser generados cuando el foco es cambiado, es decir, cuando se deja de escuchar un evento necesario para escuchar otro evento. A veces, este cambio no es bueno y se llega a un estado inválido en el cual el centro del evento no es aceptado por el foco.
- Las reglas que gobiernan el foco y las transiciones del mismo suelen ser fáciles de explicar a los usuarios, pero no son fáciles de cumplir ya que requieren un conocimiento estricto de la plataforma.

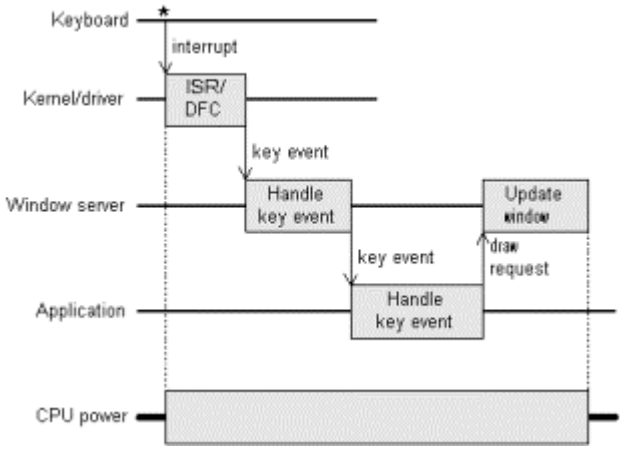


Figura 187. Diagrama de flujo de la transmisión de eventos en C++ sobre Symbian OS.

La gran mayoría de las interacciones están íntimamente relacionadas: manejador de punteros, manejador de teclado, actualización de modelos, controles de los componentes, foco, validación, etc. Sabiendo la relación que guardan todas ellas se hace sencillo desarrollar programas en Symbian OS ya que todas funcionan de forma similar, una vez que se entiende cual es el modo de funcionamiento se tiene superada la gran dificultad con la que se enfrenta un nuevo programador para este SO.

Teclado, puntero y comandos básicos: Anteriormente se ha mencionado que las funciones básicas para el control de eventos son HandlePointerEventL() y OfferKeyEventL(). Ahora se va a intentar explicar su modo de funcionamiento.

- *Manejador de eventos de teclado:* Para manejar eventos de teclado se utiliza principalmente la función OfferKeyEventL() cuyo prototipo esta a continuación:

  IMPORT_C TKeyResponse OfferKeyEventL(const TKeyEvent& aKeyEvent, TEventCode aType);

  Esta función es llamada por el controlador si el framework piensa que debe producirse un cambio en el manejador de eventos del teclado. Es necesario que el controlador puede advertir que no se esta pulsando ninguna tecla ya que hay momentos en que será muy útil conocer esta información. Es importante resaltar un par de cosas acerca de esta función:

  - Es imposible chequear nada si la función no ofrece un evento de teclado. Por ello es necesario que la función también se de cuenta de que no se esta produciendo ningún evento y pueda comunicarlo.

  - Se debe devolver un valor (EKeyWasConsumed or EKeyWasNotConsumed) para indicar exactamente que evento se ha producido.

  La función start determina el comienzo de la generación de eventos de teclado. Existen tres posibilidades – evento hardware por pulsación de tecla abajo, evento por pulsación de tecla estandar o evento hardware por pulsación de tecla arriba. Lo más interesante de los eventos de teclado estandar es el valor del segundo parámetro que se le pasa a la función EEventKey. Este parámetro se puede manejar de tres maneras diferentes:

  - Se puede ignorar si no es una tecla lo que ha generado el evento, ya que es posible reconocer quien genero el evento.

  - Se puede manejar por completo internamente, pero sin tener control ninguno del manejo. Esto ocurre cuando por ejemplo se mueve el cursor o cambia el valor interno y visual de una representación numérica en pantalla.

- Se puede manejar completamente implementando el manejador de eventos para el evento concreto producido.

El primer parámetro de OfferKeyEventL() es del tipo TKeyEvent& el cual es recogido por el manejador de eventos de teclado del servidor de ventanas y esta definido en w32std.h. UIQ define algunos códigos de teclado adicionales en quartzkeys.h. Estos representan los eventos generados por las dos o cuatro direcciones posibles del teclado (arriba, abajo, izquierda o derecha) y por la tecla de confirmación (Enter).

TEventCode es un evento de los que pueden ser recogidos por el servidor de ventanas directamente. Este se puede descomponer en subeventos que heredan del mismo que reciben por nombre EEventKey, EEventKeyDown y EEventKeyUp.

- *Manejador de eventos de puntero:* Para manejar eventos generados por el puntero se debe recurrir al uso de las dos funciones anteriormente mencionadas OfferKeyEvent() y HandlerPointerEvent(), las cuales serán llamadas por el controlador del framework cuando un puntero produzca un evento.

Generalmente el controlador de eventos de puntero necesita controlar el evento generado para decidir que hacer con el, puede manejarlo o ignorarlo como en el caso de los eventos de teclado.

Cuando se produce un evento en un puntero este siempre es enviado al controlador de eventos de puntero, no puede ser manejado por ningún otro controlador. Esta distinción con los eventos de teclado esta reflejada en la función HandlePointerEvent() ya que su tipo de retorno es void.

Al igual que en el teclado el manejador de eventos de puntero puede reaccionar a estos eventos de tres maneras diferentes:

- Ignorándolo, se puede hacer esto cuando el evento de puntero no se ha producido por utilizar un puntero que apunta al cursor abajo o cuando el evento se ha producido fuera del área visible de la ventana.

- Tratarlo internamente, cuando se produce un evento moviendo el cursor.

- O bien, generar un comando para tratar el evento fuera del controlador asignado. Esto se suele utilizar cuando el evento ocurre en un punto que no tiene un manejador de eventos concreto asignado.

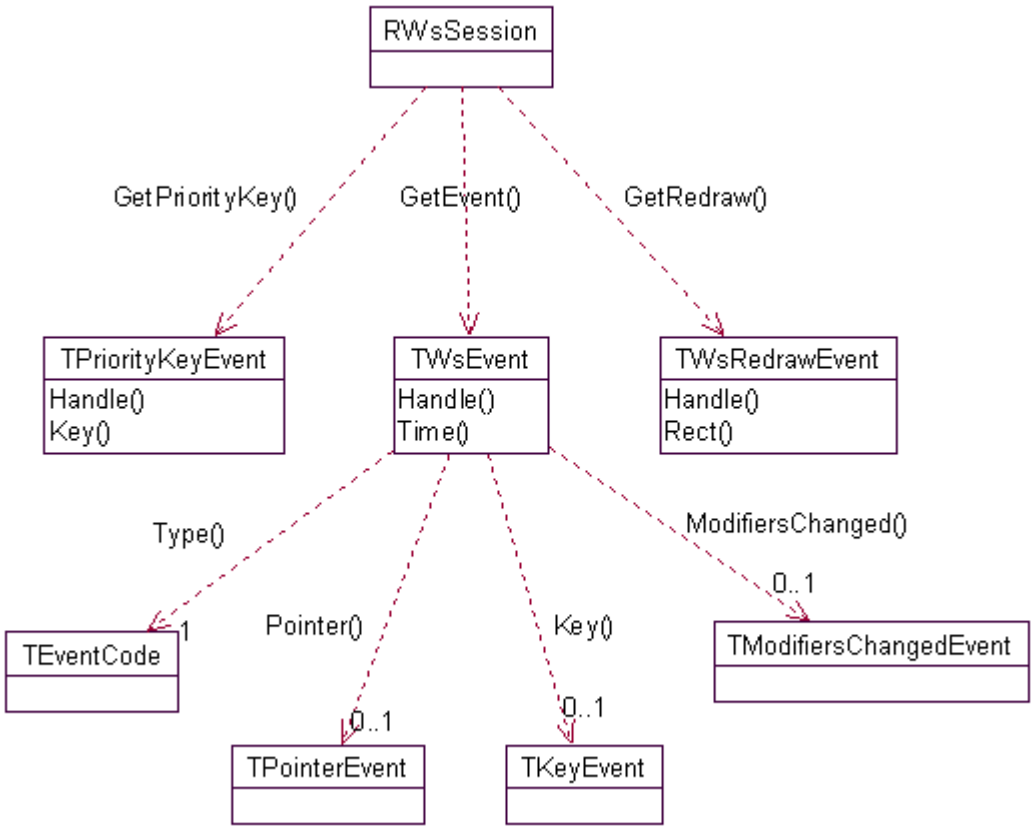


Figura 188. Diagrama de las clases que heredan del controlador de eventos de puntero.

El objeto pasado a HandlePointerEvent() recibe el nombre de TPointerEvent y esta definido en w32std.h como una estructura compuesta por una serie de botones y otras funciones que se verán a continuación:

- Botón 1, abajo (pen down);
- Botón 2, arriba (pen up).
- Otros botones, esta opción no viene definida en los dispositivos UIQ.
- Drag, normalmente no se utiliza en Symbian OS pero fuera de el es muy usada.
- Move, esta opción a pesar de ser soportada por la gran mayoría de los dispositivos no puede ser utilizada ya que tiene un gran consumo de recursos al mover las ventanas dentro de la pantalla de un MID (nunca salta este evento).
- Button repeat, esta es generada cuando se esta pulsando un botón continuamente en el mismo punto del programa.
- Switch-on: es soportada por muy pocos dispositivos, y es un evento que salta cuando una aplicación cambia de dispositivo en medio de su ejecución.
- Modifiers: este evento es similar al visto anteriormente, con el teclado. Salta cuando se pulsa una conjunción de teclas especificas (exactamente las mismas y con el mismo resultado que antes, recordemos Ctrl + Alt + Shift +…).

- *Convertir eventos en comandos:* Es posible interactuar con el controlador usando el teclado o los punteros, por ello se pueden generar comandos que controlen los eventos en diferentes puntos del programa.

  Para la generación de este tipo de comandos existen varias interfaces distribuidas por internet y con ellas, simplemente añadiéndolas al código en cuestión se puede hacer uso de esta herramienta que proporciona Symbian OS.

  Este tipo de interfaz es muy utilizada para encapsular las interacciones ocurridas entre el controlador y el resto del programa.

  Este tipo de práctica es muy útil en Symbian OS y es conveniente adoptarla lo antes posible ya que reducirá mucho el tiempo de implementación de las aplicaciones. En Uikon es una herramienta fundamental este tipo de encapsulamientos.

  En HandleCommandL(), se utiliza MEikMenuObserver para la barra de menú, así no es necesario conocer la amplia API de CEikAppUi.

  Para la barra de botones se utiliza una interfaz llamada MEikCommandObserver e igualmente para los comandos de los botones es muy utilizada la interfaz MCoeControlObserver, la cual convierte eventos de botones en comandos para las aplicaciones.

4.4.3.4 Implementación de una aplicación a modo de ejemplo.

En este punto se va a mostrar el código de una aplicación usando todos lo visto en este capitulo y en capítulos anteriores.

Se desarrollará una aplicación Hola Mundo avanzada, en la cual se mostrará el hola mundo en pantalla, pero se usarán cuadros de dialogo, desplegables con las opciones implementadas para el uso de los menús en los cuales se podrá elegir entre varias opciones disponibles, se implementaran el uso de los botones para salir de la aplicación y para poder moverse dentro de ella.

También se hace uso de multihilo y de control de eventos. A continuación se irán poniendo las distintas clases de la aplicación con sus códigos y después se añadirán algunas imágenes con la emulación de la aplicación en pantalla.

Clase cHolaActivoApp: Esta es la clase principal de la aplicación (main). En ella se encuentran las funciones obligatorias para que la aplicación pueda ser usada.

El constructor devuelve la aplicación completa, también contiene las funciones a exportar y E32Main.

```
#include "cHolaActivoApp.h"
#include <eikstart.h>
#include "cHolaActivodoc.h"
#include "holaActivoCons.h"

TUid CHolaActivoApp::AppDllUid() const
{
 return KUidHolaActivoApp;
}

CApaDocument* CHolaActivoApp::CreateDocumentL()
/**
 Sobrescribo CEikApplication para crear una instancia
 del objeto del documento
 */
{
 return CHolaActivodoc::NewL(*this);
}

EXPORT_C CApaApplication* NewApplication()
//Exporto las funciones creadas en una nueva instancia de una
//aplicación objeto.
{
 return new CHolaActivoApp;
}

GLDEF_C TInt E32Main()
/**
 Punto de entrada estandar de la función para las aplicaciones de la
 UI.
 Devuelve el código desde EikStart::RunApplication.
 */
{
 return EikStart::RunApplication(NewApplication);
}
```

Clase cHolaActivoappUI: Esta clase es usada para la creación de la vista y para añadir esta al framework.

```
#include "cHolaActivoappui.h"
#include "cHolaActivoappVista.h"


void CHolaActivoappUI::ConstructL()
 {
 //LLamada a ConstructL para inicializar los valores estandar.
 CQikAppUi::ConstructL();

 // Creo la vista y la añado al framework
 CHolaActivoappVista* appView = CHolaActivoappVista::NewLC(*this);
 iAppView = appView;
 CleanupStack::Pop(appView);

 AddViewL(*iAppView);
 }

CHolaActivoappUI::~CHolaActivoappUI(){}
```

Clase cHolaActivoappVista: En esta clase se define completamente la vista y se hace el manejo de todos los eventos que pueda usar. Como se puede comprobar necesita incluir a todas las librerías del resto de las clases para poder formar la interfaz gráfica de la aplicación. El código esta comentado así que se supone que no habrá problemas para su comprensión.

```
#include "CHolaActivoappVista.h"
#include <QikCommand.h>
#include <HolaActivo.rsg>
#include "CFlashHola.h"
#include "DelHola.h"
#include "CHolaMultiP.h"
#include "HolaActivoCons.h"
#include "HolaActivo.hrh"


CHolaActivoappVista::CHolaActivoappVista(CQikAppUi& aAppUi)
:CQikViewBase(aAppUi, KNullViewId),
iCommandManager(CQikCommandManager::Static())
 {
 }

CHolaActivoappVista::~CHolaActivoappVista()
/**
Destructor para la vista
*/
 {
 // Defino la flag para prevenir que se redibuje la vista
 // durante su destrucción.
 iAmBeingDestroyed = ETrue;

 delete iDelayedHello;
 delete iFlashingHello;
 delete iMultiHello;
 delete iText;
 }

void CHolaActivoappVista::ConstructL()
 {
 // Llamo a ConstructL para incializarlo con los valores estandar.
 CQikViewBase::ConstructL();

 // Creo los objetos activos
 iDelayedHello = DelHola::NewL();
 iFlashingHello = CFlashHola::NewL(*this);
 iMultiHello = CHolaMultiP::NewL(*this);

 // Creo el mensaje de texto leyendo los contenidos a partir
 // del archivo de resolución.
    iText = iEikonEnv->AllocReadResourceL(R_HOLAACTIVO_TEXT_HELLO);

 // El estado inicial indica que se esta viendo correctamente el
 //mensaje de texto.
 iShowText = ETrue;
 }

CHolaActivoappVista* CHolaActivoappVista::NewLC(CQikAppUi& aAppUi)
 {
 //Creo e inicializo la vista.
```

```
      CHolaActivoappVista* self = new(ELeave)
CHolaActivoappVista(aAppUi);
      CleanupStack::PushL(self);
      self->ConstructL();
      return self;
      }

     void CHolaActivoappVista::ViewConstructL()
      {
      // Cargo la información de la configuración del UI.
      ViewConstructFromResourceL(R_HOLAACTIVO_UI_CONFIGURATIONS);
         InitComponentArrayL();
      }

     TVwsViewId CHolaActivoappVista::ViewId()const
     //Esta función devuelve el ID.
      {
      return TVwsViewId(KUidHolaActivoApp, KUidHolaActivoAppView);
      }

     void CHolaActivoappVista::SizeChanged()
         {
         }

     void CHolaActivoappVista::Draw(const TRect&) const
         {
      CWindowGc& gc = SystemGc();
      gc.Clear();
      TRect rect=Rect();
      rect.Shrink(10,10);
      gc.DrawRect(rect);

      if (iShowText)
            {
            rect.Shrink(1,1);
            const CFont* font=iEikonEnv->TitleFont();
            gc.UseFont(font);
            const TInt baseline = rect.Height()/2 + font-
            >AscentInPixels()/2;
            gc.DrawText(*iText, rect, baseline,
CGraphicsContext::ECenter);
            gc.DiscardFont();
            }
      else
            {
            rect.Shrink(1,1);
            gc.SetPenStyle(CGraphicsContext::ENullPen);
            gc.SetBrushStyle(CGraphicsContext::ESolidBrush);
            gc.DrawRect(rect);
            }
         }

     void CHolaActivoappVista::HandleCommandL(CQikCommand& aCommand)
     /*
      * Manejo los eventos generados por los comandos en la vista.
      * Llamo al UI framework cuando un comando salta.
      * Las Ids de los comandos estan definidos en el archivo .hrh
     */
      {
      switch(aCommand.Id())
            {
```

```
        case EHolaActivoCmdSetHello:
              {
              iDelayedHello->Cancel();
              iDelayedHello->SetHello(3000000); // Retraso de 3 seg.
              break;
              }
        case EHolaActivoCmdCancelHello:
              {
              iDelayedHello->Cancel();
              iEikonEnv->InfoMsg(R_HOLAACTIVO_TEXT_CANCELLED);
              break;
              }
        case EHolaActivoCmdStartFlashing:
              {
              iFlashingHello->Cancel();
              iFlashingHello->Start(1000000); // 1 segundo
              iEikonEnv->InfoMsg(R_HOLAACTIVO_TEXT_STARTED);
              break;
              }
        case EHolaActivoCmdStopFlashing:
              {
              iFlashingHello->Cancel();
              iEikonEnv->InfoMsg(R_HOLAACTIVO_TEXT_STOPPED);
              break;
              }
      case EHolaActivoCmdStartMultiHello:
              {
             iFlashingHello->Cancel();
             iMultiHello->Cancel();
             iMultiHello->Start(3000000); // 1 segundo
             iEikonEnv->InfoMsg(R_HOLAACTIVO_TEXT_MULTI_STARTED);
             break;
              }
      case EHolaActivoCmdCancelMultiHello:
              {
             iMultiHello->Cancel();
             iEikonEnv->InfoMsg(R_HOLAACTIVO_TEXT_MULTI_CANCEL);
             break;
              }

        // Para Volver y Salir de la aplicación se maneja el evento
        //con CQikViewBase.
        default:
              CQikViewBase::HandleCommandL(aCommand);
              break;
        }
 }

void CHolaActivoappVista::ShowText(TBool aShow)
    {
    iShowText = aShow;
    if (iAmBeingDestroyed == EFalse)
     {
     DrawNow();
     }
    }

TBool CHolaActivoappVista::IsTextShowing() const
    {
    return iShowText;
    }
```

Clase cHolaActivodoc: Se usa para crear una UI para la aplicación. La última función es la encargada de devolver la UI creada al framework.

```
#include "CHolaActivodoc.h"

#include <QikApplication.h>
#include "CHolaActivoappUI.h"

CHolaActivodoc* CHolaActivodoc::NewL(CQikApplication& aApp)

//Inicializo los objetos,
 {

 CHolaActivodoc* self = new (ELeave) CHolaActivodoc(aApp);

 CleanupStack::PushL(self);
 self->ConstructL();

 CleanupStack::Pop(self);

 return self;
 }

CHolaActivodoc::CHolaActivodoc(CQikApplication& aApp)

/**
 Este constructor es definido para inicializar la plataforma
 especifica de la super clase.
 */
:CQikDocument(aApp)
 {

    // No hace falta implementar
 }

void CHolaActivodoc::ConstructL()

//Aqui comienza la segunda fase de construccion del modelo.
 {

    // No hace falta implementar
 }

CEikAppUi* CHolaActivodoc::CreateAppUiL()
/**
 Implemetancion de CEikDocument para crear una nueva UI
 para la aplicación.

 Esta función devuelve la nueva app UI del objeto.
 */

 {

 return new (ELeave) CHolaActivoappUI;

 }
```

Clase delHola: Esta nueva clase es utilizada para leer información de un archivo rsg. Con ella se crea un delay que nos servirá para mostrar la aplicación correctamente.

```
#include "delHola.h"
#include <eikenv.h>
#include <HolaActivo.rsg>

DelHola* DelHola::NewL()
 {
 DelHola* self = new (ELeave) DelHola();
 CleanupStack::PushL(self);
 self->ConstructL();
 CleanupStack::Pop(self);
 return self;
 }

DelHola::DelHola()
: CActive(CActive::EPriorityStandard)
 {
 CActiveScheduler::Add(this);
 }

void DelHola::ConstructL()
 {
 iEnv = CEikonEnv::Static();
 User::LeaveIfError(iTimer.CreateLocal());
 }

DelHola::~DelHola()
 {
 Cancel();
 iTimer.Close();
 }
// respuesta

void DelHola::SetHello(TTimeIntervalMicroSeconds32 aDelay)
 {
 _LIT(KDelayedHelloPanic, "DelHola");
 __ASSERT_ALWAYS(!IsActive(), User::Panic(KDelayedHelloPanic, 1));

 iTimer.After(iStatus, aDelay);
 SetActive();
 }
// desde CActive

void DelHola::RunL()
 {
 iEnv->InfoMsg(R_HOLAACTIVO_TEXT_HELLO);
 }

void DelHola::DoCancel()
 {
 iTimer.Cancel();
 }

TInt DelHola::RunError(TInt /*Error*/)
 {
 return KErrNone;


 }
```

Clase cFlashHola: En esta clase se crea la animación flash que forma parte de nuestra aplicación. Esta animación es devuelta para su uso en la misma.

```
#include "cFlashHola.h"
#include "cHolaActivoappVista.h"


// Constructor/Destructor

CFlashHola* CFlashHola::NewL(CHolaActivoappVista& aAppView)
 {
 CFlashHola* self=new(ELeave) CFlashHola(aAppView);
 CleanupStack::PushL(self);
 self->ConstructL();
 CleanupStack::Pop(self);
 return self;
 }

CFlashHola::CFlashHola(CHolaActivoappVista& aAppView)
: CActive(CActive::EPriorityStandard), iAppView(aAppView)
 {
 CActiveScheduler::Add(this);
 }

void CFlashHola::ConstructL()
 {
 User::LeaveIfError(iTimer.CreateLocal());
 }


CFlashHola::~CFlashHola()
 {

 Cancel();
 iTimer.Close();
 }

//Respuesta

void CFlashHola::Start(TTimeIntervalMicroSeconds32 aHalfPeriod)
 {

 _LIT(KFlashingHelloPeriodPanic, "CFlashHola");
 __ASSERT_ALWAYS(!IsActive(), User::Panic(KFlashingHelloPeriodPanic,
 1));

 // Recuerda una parte del periodo
 iHalfPeriod=aHalfPeriod;

 // Muestra el texto para que la aplicación comience con el
 ShowText(EFalse);

 iTimer.After(iStatus, iHalfPeriod);
 SetActive();
 }

// desde CActive
```

```
void CFlashHola::RunL()
 {

 // Cambio la visibilidad de la vista de la aplicación.
 const TBool textIsShowing = iAppView.IsTextShowing();
 ShowText(!textIsShowing);

 // Respuesta
 iTimer.After(iStatus, iHalfPeriod);
 SetActive();
 }

void CFlashHola::DoCancel()
 {

 // Muestra el texto
 ShowText(ETrue);

 // Apago el timer
 iTimer.Cancel();
 }

TInt CFlashHola::RunError(TInt /*Error*/)
 {

 return KErrNone;
 }

void CFlashHola::ShowText(TBool aShowText)
 {

 iAppView.ShowText(aShowText);
 }
```

Clase cHolaMultiP: Esta es la clase utilizada para generar el multihilo, con esta clase genero dos aplicaciones que se ejecutan concurrentemente, una el hola mundo anterior y esta concurre con otro hola mundo generado en un botón que se sitúa en la parte superior derecha de la pantalla.

```
#include "cHolaMultiP.h"

#include <eikenv.h>
#include <HolaActivo.rsg>
#include "CHolaActivoappVista.h"


CHolaMultiP::CHolaMultiP(CHolaActivoappVista& aAppView)
: CActive(CActive::EPriorityStandard), iAppView(aAppView)
 {
 CActiveScheduler::Add(this);
 }

CHolaMultiP::~CHolaMultiP()
 {
 Cancel();

 iTimer.Close();
 }
```

```
void CHolaMultiP::ConstructL()
 {
 User::LeaveIfError(iTimer.CreateLocal());

 iEnv = CEikonEnv::Static();
 }

CHolaMultiP* CHolaMultiP::NewL(CHolaActivoappVista& aAppView)
 {
 CHolaMultiP* self = new (ELeave) CHolaMultiP(aAppView);
 CleanupStack::PushL(self);
 self->ConstructL();
 CleanupStack::Pop(self);
 return self;
 }

void CHolaMultiP::CompleteSelf()
 {
 TRequestStatus* pStat = &iStatus;
 User::RequestComplete(pStat, KErrNone);
 SetActive();
 }

void CHolaMultiP::Start(TTimeIntervalMicroSeconds32 aDelay)
 {
 _LIT(KMultiPartHelloPanic, "CHolaMultiP");
 __ASSERT_ALWAYS( iState == EIdle, User::Panic(KMultiPartHelloPanic,
 1));

 iDelay = aDelay;
 iState = EShowHello;
 CompleteSelf();
 }

void CHolaMultiP::ShowText(TBool aShowText)
 {
 iAppView.ShowText(aShowText);
 }

void CHolaMultiP::RunL()
 {
 THelloState nextState = iState;

    switch( iState )
        {
 case EShowHello:
        {
        ShowText(ETrue);

        // Respuesta
        iTimer.After(iStatus, iDelay);
        SetActive();

        nextState = EHideHello;
        }
         break;
 case EHideHello:
        {
        ShowText(EFalse);
```

```
        CompleteSelf();
        nextState = EShowInfoHello;
        }
         break;
  case EShowInfoHello:
        {
        iEnv->InfoMsg(R_HOLAACTIVO_TEXT_HELLO);

        CompleteSelf();
        nextState = EWaitState;
        }
         break;
  case EWaitState:
        {
        // Respuesta 2
        iTimer.After(iStatus, iDelay);
        SetActive();

        nextState = EShowHello;
        }
         break;
  default:
        break;
        }

     iState = nextState;
  }

void CHolaMultiP::DoCancel()
  {
  switch( iState )
        {
  case EHideHello:
  case EShowHello:
        {
        iTimer.Cancel();
        }
         break;
  default:
        break;
        }

     ShowText(ETrue);
  iState = EIdle;
  }

TInt CHolaMultiP::RunError(TInt /*Error*/)
     {
     iState = EIdle;
     ShowText(ETrue);
     return KErrNone;
     }
```

Una vez vistos los códigos de la aplicación se pasará a poner unas imágenes de la emulación.

4.4.3.5 Emulación de la aplicación con Carbide C++.

A continuación se incluyen unas imágenes de la emulación de la aplicación con el Carbide C++.

Figura 189. Captura de pantalla del emulador de Carbide C++. Esta es la pantalla de inicio de la aplicación, cuando la arrancas entra en ella.

Figura 190. Captura de pantalla del emulador de Carbide C++. Esta imagen es del menú de la aplicación. Como se puede ver tiene diversas opciones entre las que se puede destacar inicializar el flash o el multi-hola.

Figura 191. Esta pantalla pertenece a la opción Inicializar multi-hola.

Figura 192. Esta pantalla es una captura de la aplicación funcionando en modo multi-hola en concreto en el momento en que muestra el botón del margen superior derecho.

4.4.4 Comunicaciones e Intercambio con Symbian OS

4.4.4.1 Introducción

En este capitulo se va a ver la comunicación e intercambio de archivos con Symbian OS. Este es similar al desarrollado para J2ME en el apartado 3.3.3. En ese apartado ya se explicaban las diversas tecnologías disponibles y había una especificación de las características de cada una así que en este capitulo todo ira centrado directamente sobre Bluetooth y el desarrollo de una aplicación utilizando las herramientas que provee C++ para su diseño.

Se va a programar una aplicación que sea capaz de establecer una conexión a través de bluetooth, y enviar datos en ambas direcciones. Para ello se va a presentar la clase BluetoothM. La aplicación a desarrollar se basara en el concepto de cliente-servidor e integrara todos los métodos necesarios para poner a la escucha el servidor, conectar un cliente y enviar/recibir datos.

Se compondrá principalmente de dos archivos: BluetoothM.cpp y BluetoothM.h. Correspondientes al código fuente de la aplicación y a las cabeceras respectivamente.

Figura 193. Logotipo de la tecnología Bluetooth.

4.4.4.2 La clase BluetoothM

Esta clase esta compuesta por un conjunto de funciones definidos en la cabecera (BluetoothM.h). Las funciones que la componen son las siguientes:

- BluetoothM(BluetoothMObserver *obs);
- BluetoothM();
- void Listen();
- void Connect();
- void Send(TBuf8 &data, bool force);
- void Close();
- bool SendingBusy();
- bool AskDevice(TBTDeviceResponseParamsPckg &result);
- EBTState state;

Ahora se comentarán cada una de las funciones y la utilidad que tiene cada una dentro de la aplicación.

- *BluetoothM(BluetoothMObserver *obs):* Este método es el constructor de la clase. Toma como parámetro un puntero a un objeto del tipo BluetoothMObserver, pertenecer a los controladores de Avkon.

  La clase que intente comunicarse con BluetoothM (recibir notificaciones cuando llegue un paquete, o cuando se haya creado una conexión, etc.) debe heredar de la clase BluetoothMObserver y sobrecargar los métodos de la misma para poder recibir las notificaciones. Esta clase también esta definida en BluetoothM.h y tiene la siguiente forma:

```
class BluetoothMObserver {
public:
virtual void BluetoothIncomingData(TBuf8 &data) = 0;
virtual void BluetoothConnRequest() = 0;
virtual void BluetoothConnAcepted() = 0;
};
```

  Se compone de tres métodos de los cuales se va a ver su cometido a continuación. El método *BluetoothIncomingData* será llamado cuando lleguen datos a través de la conexión, se le debe pasar un parámetro con los datos que le llegan. *BluetoothConnRequest* será llamado cuando a una aplicación que esté actuando como servidor se le haya conectado un cliente. *BluetoothConnAcepted* será llamado cuando a una aplicación que esté actuando como cliente y haya efectuado una petición de conexión se le haya aceptado esa petición.

  De esta manera, la aplicación tendría una forma similar a esto:

```
Class MyApp : public BluetoothMObserver {
public:
//Constructor
MyApp();

void BluetoothIncomingData (TBuf8 &data);
void BluetoothConnRequest ();
void BluetoothConnAcepted ();

//Resto de métodos o atributos de la clase
//....
//....
private:
BluetoothM * bluetoothM;
};
```

Y su implementación sería:

```
MyApp::MyApp(){


//Construimos el objeto BluetoothM, y le pasamos nuestro
//puntero por parámetro para poder "observar las
//notificaciones que lleguen.

bluetoothM = new BluetoothM(this);

}

void MyApp:: BluetoothIncomingData (TBuf8 &data){
//Acaban de llegar datos, los cuales nos llegan a través del
//parámetro data procesarlos o hacer las tareas pertinentes
}

void MyApp:: BluetoothConnRequest (){


//La aplicación estaba actuando como servidor y acabamos de
//recibir una conexión. Ya podríamos empezar a enviar datos
//si quisiéramos ya que ya hay creado un canal de conexión.

}

void MyApp:: BluetoothConnAcepted (){


//La aplicación estaba actuando como servidor y nuestra
//petición de conexión ha sido aceptada. Ya podemos empezar a
//enviar datos si queremos.
}
```

Con esto quedaría implementada la parte de creación del objeto BluetoothM además de la comunicación con los distintos eventos entre esta y la aplicación. Ahora es momento de retomar las explicaciones de como funcionan el resto de métodos de BluetoothM.

- *void Listen():* Este método es utilizado solo por las aplicaciones que actuan como servidor. Este método es muy utilizado por todos los servidores, se utiliza para poder recibir una conexión.

  Una vez que el servidor tiene asociado a un cliente, este método se utilizará para hacer de escuchador de eventos del mismo.

- *void Connect():* Se utiliza para realizar la petición de conexión a una aplicación. Cuando se invoca a este método, aparece una ventana preguntando si se quieren buscar mas dispositivos o elegir uno de entre los que aparecen en la lista.

  Una vez seleccionado el terminal con el que se quiere conectar, la clase BluetoothM iniciará todo el proceso de establecimiento de la conexión. Una vez establecida la conexión el metodo BluetoothConnAcepted avisará de ello.

- *void Send(TBuf8 &data, bool force):* Este método entra en funcionamiento una vez que la conexión ha sido establecida. Se utiliza para enviar datos hacia el otro lado de la conexión.

  El primer parametro que toma sirve para codificar la información que se quiere enviar byte a byte.

  El Segundo parametro sirve para el controlar el acceso al medio. Si se realizan dos llamadas a Send consecutives (con pocos milisegundos de diferencia entre ellas) es possible que cuando se ejecute la segunda llamada la primera no haya sido cursada todavia. Con este parametro se puede controlar este tipo de cosas. Los dos posibles estados en los que se puede poner este parametro son los siguientes:

  - true: si el BluetoothM esta ocupado con un envió anterior cuando es llamado de nuevo Send, este primer envio es eliminado y la segunda llamada tendrá prioridad sobre la anterior.

  - false: si el BluetoothM esta ocupado con un envío anterior cuando de nuevo se llama a Send, el Segundo envío sera rechazado y deberá realizarse de nuevo un poco mas tarde.

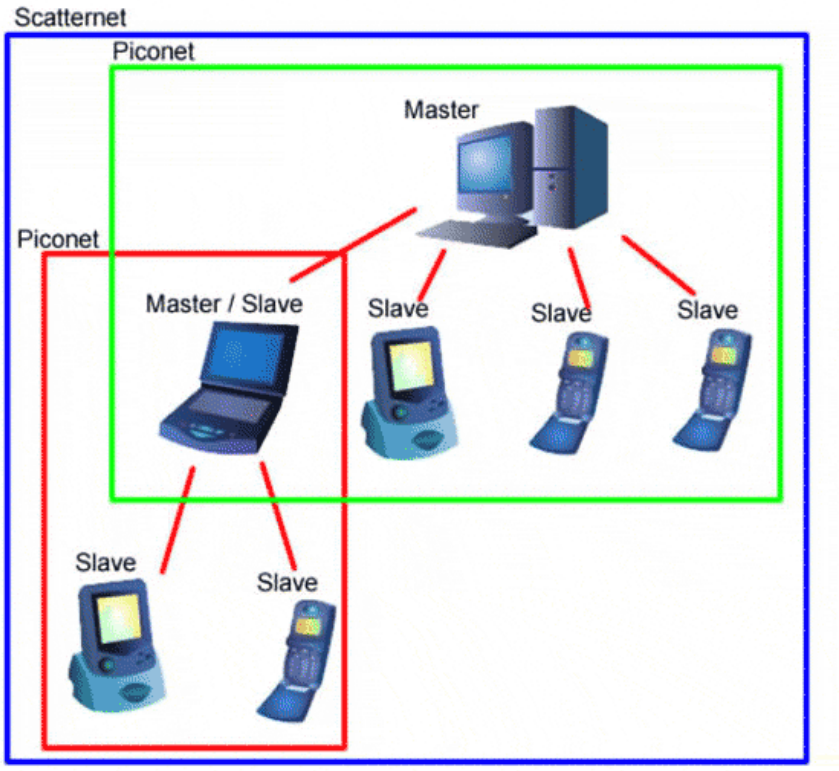


Figura 194. Esquema de una red Bluetooth formada por distintos dispositivos.

Para comprobar la situación explicada anteriormente, es decir, si el BluetoothM esta ocupado con un envío en ese instante, esta el método *SendingBusy*, que devolverá true si esta ocupado, o false en caso contrario. Esta situación de ocupación se da muy raramente pero es importante tenerla en cuenta para evitar problemas con los Active Objects.

- *void Close():* Se utiliza para cerrar una conexión activa.

- *bool AskDevice(TBTDeviceResponseParamsPckg &result):*Con este método se puede invocar manualmente a la ventana que busca dispositivos dentro del alcance.

- *EBTState state:* Este es un atributo público que permite saber en que estado se encuentra en un momento dado el BluetoothM. La definición del tipo del atributo (EBTState) esta definido en BluetoothM y su definición es la siguiente:

```
enum EBTState{
eDisconnected,
eListening,
eConnecting,
eConnected,
};
```

Con esto ya esta visto todo el funcionamiento de la clase y explicado como puede crearse una aplicación cliente servidor para enviar datos entre ella a través del bluetooth.

A la hora de querer usar esta clase en una aplicación distinta de la que aquí se va a explicar, lo primero que se deberá hacer es añadir las librerías necesarias para que compile. Para esto es necesario ir al archivo de especificación (mmp) y añadir las siguientes líneas:

```
LIBRARY bluetooth.lib
LIBRARY btmanclient.lib
LIBRARY BTExtNotifiers.lib
LIBRARY SdpAgent.lib
LIBRARY SdpDatabase.lib
LIBRARY esock.lib
```

Una vez hecho esto, se deberá añadir al proyecto nuevo los ficheros BluetoothM.h y BluetoothM.cpp. A partir de esta explicación se desarrollará ahora una aplicación utilizando estas clases.

4.4.4.3 Aplicación usando la clase BluetoothM

Esta aplicación tiene un funcionamiento sencillo, se trata de enviar y recibir coordenadas a través del bluetooth del terminal para mover dos bolitas que estarán en la pantalla. No se podrán mover las dos bolas a la vez, ni manejar las dos bolas con el mismo terminal. Al realizar la emulación se necesitan dos pantallas ya que una se encargará de mover una bola y la otra de la otra.

Uno de los terminales permanecerá en espera (en modo escucha “Listen”) mientras que el otro modifica la posición de la bola y una vez terminada la operación el control de acceso se le otorgará al otro terminal que podrá realizar lo mismo con su bola respectiva.

Código de la Aplicación:

Clase BluetoothM.h

```
#ifndef __BLUETOOTHM_H__
#define __BLUETOOTHM_H__

#include <es_sock.h>
#include <btextnotifiers.h>
#include <btsdp.h>

#define BT_PACKET_SIZE 256

//Posibles estados
enum EBTState{
 eDisconnected,
 eListening,
 eConnecting,
 eConnected,
};

//Forward declarations
class BluetoothM;

class BluetoothMObserver{
public:

 virtual void BluetoothIncomingData(TBuf8<BT_PACKET_SIZE> &data) =
 0;
 virtual void BluetoothConnRequest() = 0;
 virtual void BluetoothConnAcepted() = 0;
};

class BTSender : CActive {
public:

 BTSender(BluetoothMObserver *bto,BluetoothM *_manager);
 void SendData(TBuf8<BT_PACKET_SIZE> &data);
 void SetSocket(RSocket *s);
 void CancelRequest();
 bool Busy();

private:

 BluetoothMObserver *parent;
 BluetoothM *manager;
 bool sending;

 void RunL();
 void DoCancel();

 RSocket *socket;
};

class BTReceiver : CActive {
public:

 BTReceiver(BluetoothMObserver *bto,BluetoothM *_manager);
 void RequestData();
 void SetSocket(RSocket *s);
```

```
 void CancelRequest();

private:
 BluetoothMObserver *parent;
 BluetoothM *manager;

 RSocket *socket;
 TBuf8<BT_PACKET_SIZE> buffer;
 TSockXfrLength transferLength;

 void RunL();
 void DoCancel();
};

class BluetoothM : public CActive,public MSdpAgentNotifier,public
MSdpAttributeValueVisitor {

public:

 BluetoothM(BluetoothMObserver *obs);
 ~BluetoothM();

 void Listen();
 void Connect();
 void Send(TBuf8<BT_PACKET_SIZE> &data,bool force);
 void Close();
 bool SendingBusy();

 bool AskDevice(TBTDeviceResponseParamsPckg &result);

 EBTState state;

protected:

 void DoCancel();
 void RunL();

private:

 void FindService(const TBTDevAddr &addr);

 //De MSdpAgentNotifier
 void NextRecordRequestComplete(TInt aError, TSdpServRecordHandle
 aHandle, TInt aTotalRecordsCount);

 void AttributeRequestComplete(TSdpServRecordHandle aHandle, TInt
 aError);

 void AttributeRequestResult(TSdpServRecordHandle aHandle,
 TSdpAttributeID aAttrID, CSdpAttrValue* aAttrValue);

 //De MSdpAttributeValueVisitor
 void VisitAttributeValueL(CSdpAttrValue &aValue, TSdpElementType
 aType);

 void StartListL(CSdpAttrValueList &aList);
 void EndListL();

 void StartAdvertising(int channel);
 void UpdateAvailability(bool b);
 void SetSecurityOnChannel(int channel);
```

```
 BluetoothMObserver *observer;
 BTSender sender;
 BTReceiver receiver;

 RSocketServ iSocketServer;
 RSocket iListeningSocket;
 RSocket iAcceptedSocket;
 RSocket iSendingSocket;

//

 TBTDeviceResponseParamsPckg deviceFound;
 TInt remotePort;
 CSdpAgent *sdpAgent;
 CSdpSearchPattern *searchPattern;
 TInt iSdpCurrentRecordCount;
 TInt iSdpTotalRecordCount;


 TSdpServRecordHandle recordAdvertiser;
 int recordState;
 RSdp iSdpSession;
 RSdpDatabase iSdpDatabase;
 bool sdpSessionConnected;
};

#endif
```

Clase BluetoothM.cpp

```
#include <bt_sock.h>
#include <btsdp.h>
#include <eikenv.h>
#include <aknnotewrappers.h>
#include <btmanclient.h>
#include "BluetoothM.h"

BluetoothM::BluetoothM(BluetoothMObserver
*obs):CActive(CActive::EPriorityStandard),sdpAgent(NULL),

searchPattern(NULL),observer(obs),sender(obs,this),

receiver(obs,this),recordAdvertiser(0){
        sdpSessionConnected = false;

 //Inicialmente nuestro gestor de bluetooth esta desconectado
 state = eDisconnected;
 CActiveScheduler::Add(this);
}

BluetoothM::~BluetoothM(){
 //Cuando se destruye el objeto, debemos cancelar todas las
 //peticiones que tuvieran activas tanto el objeto activo que
 //utilizamos para enviar, como el que utilizamos para recibir
 sender.CancelRequest();
 receiver.CancelRequest();
 Cancel();
```

```
}

void BluetoothM::Listen(){
 //Modo servidor
 TInt err;

 //Comprobamos que estemos desconectados, si no la llamada de este
 //metodo no hace nada.Deberiamos hacer un Close primero y despuer
 //llamar a Listen para eso.

 if(state == eDisconnected){
       iSocketServer.Connect();
       //Abrimos un socket y comprobamos que no haya ningun problema
       if((err = iListeningSocket.Open(iSocketServer,_L("RFCOMM")))
       != KErrNone){
             iSocketServer.Close();
             return;
       }

       TInt channel;
       //Obtenemos un puerto libre por el que poner a la escucha el
       //socket
       iListeningSocket.GetOpt(KRFCOMMGetAvailableServerChannel,
       KSolBtRFCOMM, channel);

       TBTSockAddr listeningAddress;
       //Le asignamos el puerto a la direccion y nos ponemos a la
       //escucha
       listeningAddress.SetPort(channel);
       iListeningSocket.Bind(listeningAddress);
       iListeningSocket.Listen(1);

       //Cerramos el socket por el que vamos a establecer la
       //conexion con el cliente por si estuvise abierto de una
       //conexion anterior , y lo reabrimors.
        iAcceptedSocket.Close();
       iAcceptedSocket.Open(iSocketServer);

       //El BluetoothM pasa al estado "a la escucha"
       state = eListening;

       //Asociamos el iAcceptedScoket como el socket que vamos a
       //usar cuando se acepte la conexion
       iListeningSocket.Accept(iAcceptedSocket, iStatus);

       //Asociamos al sender y al receiver al socket anterior, para
       //poder enviar y recibir datos a traves de el
       sender.SetSocket(&iAcceptedSocket);
       receiver.SetSocket(&iAcceptedSocket);

       //Este metodo debe ser llamado una vez estemos a la escucha
       //para que el resto de dispositivos pueda "ver" nuestra
       //conexion activa
        StartAdvertising(channel);
        UpdateAvailability(true);
        SetSecurityOnChannel(channel);

        SetActive();
 }
}
```

```
void BluetoothM::Connect(){
 //Modo cliente
 if((state == eDisconnected) && !IsActive()){
//Si estamos desconectados y no hay ninguna peticion en curso,
//llamamos a AskDevice con lo que saldra el cuadro de dialogo de
//dispositivos bluetooth en pantalla.Si el usuario seleccion uno
//le pasamos los datos del seleccionado a la funcion FindService
       if(AskDevice(deviceFound)){
             FindService(deviceFound().BDAddr());
       }
 }
}

void BluetoothM::Send(TBuf8<BT_PACKET_SIZE> &data,bool force){
 if(sender.Busy()){
       if(force){
       //Si el sender esta ocupado y aun asi queremos enviar datos,
       //cancelamos la peticion actual y realizamos la nuestra
             sender.CancelRequest();
             sender.SendData(data);
       }
 }else{
       sender.SendData(data);
 }
}

void BluetoothM::Close(){
 DoCancel();
}

bool BluetoothM::SendingBusy(){
 return sender.Busy();
}

bool BluetoothM::AskDevice(TBTDeviceResponseParamsPckg &result){
 bool ret = true;

 // Crear el notifier
 RNotifier not;
 not.Connect();
 // Seleccion del servicio
 TBTDeviceSelectionParams selectionParams;
 TUUID targetServiceClass(0x1101);//Serial
 selectionParams.SetUUID(targetServiceClass);
 TBTDeviceSelectionParamsPckg pckg(selectionParams);

 TRequestStatus status;
 not.StartNotifierAndGetResponse(status,KDeviceSelectionNotifierUid,
 pckg,result);

 User::WaitForRequest(status);
 // Extraer el nombre del dispositivo seleccionado
 if(!result().IsValidBDAddr()){
       //Seleccion invalida
       ret = false;
 }
  // Limpieza
 not.CancelNotifier(KDeviceSelectionNotifierUid);
 not.Close();
 return ret;
}
```

```
void BluetoothM::FindService(const TBTDevAddr &addr){
 if(sdpAgent) delete sdpAgent;
 sdpAgent = CSdpAgent::NewL(*this,addr);

 if(searchPattern) delete searchPattern;
 searchPattern = CSdpSearchPattern::NewL();

 iSdpCurrentRecordCount = 0;
 iSdpTotalRecordCount = 0;

 searchPattern->AddL(0x1101);
 sdpAgent->SetRecordFilterL(*searchPattern);
 sdpAgent->NextRecordRequestL();
}


void BluetoothM::NextRecordRequestComplete(TInt aError,
TSdpServRecordHandle aHandle, TInt aTotalRecordsCount){
 iSdpTotalRecordCount = aTotalRecordsCount;
 sdpAgent->AttributeRequestL(aHandle,
 KSdpAttrIdProtocolDescriptorList);
}

void BluetoothM::AttributeRequestComplete(TSdpServRecordHandle
aHandle, TInt aError){
 iSdpCurrentRecordCount++;

 if(iSdpCurrentRecordCount < iSdpTotalRecordCount){
       sdpAgent->NextRecordRequestL();
 }
 else{
       state = eConnecting;

       iSocketServer.Connect();
       iSendingSocket.Open(iSocketServer, _L("RFCOMM"));

       TBTSockAddr addr;
       addr.SetBTAddr(deviceFound().BDAddr());
       addr.SetPort(remotePort);

       iSendingSocket.Connect(addr, iStatus);

       sender.SetSocket(&iSendingSocket);
       receiver.SetSocket(&iSendingSocket);

       SetActive();
 }
}

void BluetoothM::AttributeRequestResult(TSdpServRecordHandle
aHandle, TSdpAttributeID aAttrID, CSdpAttrValue* aAttrValue){

 if (aAttrValue->Type() == ETypeDES) {
       CSdpAttrValueDES* attrValueDES = (CSdpAttrValueDES*)
       aAttrValue;
       attrValueDES->AcceptVisitorL(*this);
 }

 delete aAttrValue;
}
```

```
void BluetoothM::VisitAttributeValueL(CSdpAttrValue &aValue,
TSdpElementType aType){
 if(aValue.Type() == ETypeUint){
       remotePort = aValue.Uint();
 }
}

void BluetoothM::StartListL(CSdpAttrValueList&){
}

void BluetoothM::EndListL(){
}

void BluetoothM::RunL(){
 if(iStatus != KErrNone){
 //Gestion de errores.Ahora mismo no hay nada, habria que
 //implementar las posibles desconexiones
       switch(state){
       case eConnecting:
             break;
       }
 }
 else{
       switch(state){
       case eListening:
       //Estabamos a la escucha y hemos recibido conexion.Pasamos a
       //estado conectado.
             state = eConnected;
             //Dejamos de mostrar nuestra conexion a los demas.
             UpdateAvailability(false);
             //El receiver ya puede empezar a recibir datos si los
             //hay.
             receiver.RequestData();
             //Avisamos al observer de que se ha establecido la
             //conexion.
             observer->BluetoothConnRequest();
             break;
       case eConnecting:
             {
             //Estabamos a la espera de la aceptacion de la conexion
             //y ya se ha llevado a cabo.
             state = eConnected;
             //Avisamos al observer.
             observer->BluetoothConnAcepted();
             //Ya podemos empezar a recibir datos si los hay.
             receiver.RequestData();
             }
             break;
       }
 }
}

void BluetoothM::DoCancel(){
 //Cerramos todos los sockets
 iAcceptedSocket.Close();
 iListeningSocket.Close();
 iSendingSocket.Close();
 iSocketServer.Close();
```

```
        if(recordAdvertiser != 0){
            iSdpDatabase.DeleteRecordL(recordAdvertiser);
            recordAdvertiser = 0;
        }
        iSdpDatabase.Close();
        iSdpSession.Close();

     delete sdpAgent;
     delete searchPattern;
     state = eDisconnected;
    }

    void BluetoothM::StartAdvertising(int channel){
        if(recordAdvertiser != 0){
            iSdpDatabase.DeleteRecordL(recordAdvertiser);
            recordAdvertiser = 0;
        }
        if(!sdpSessionConnected){
            iSdpSession.Connect();
            iSdpDatabase.Open(iSdpSession);
            sdpSessionConnected = true;
        }

        iSdpDatabase.CreateServiceRecordL(0x1101, recordAdvertiser);
        CSdpAttrValueDES* vProtocolDescriptor =
     CSdpAttrValueDES::NewDESL(NULL);
        CleanupStack::PushL(vProtocolDescriptor);

        TBuf8<1> temp;
        temp.Append((TChar)channel);
          vProtocolDescriptor->StartListL()->BuildDESL()->StartListL()->
          BuildUUIDL(KL2CAP)->EndListL()->BuildDESL()->StartListL()->
          BuildUUIDL(KRFCOMM)->BuildUintL(temp)->EndListL()->EndListL();


iSdpDatabase.UpdateAttributeL(recordAdvertiser,KSdpAttrIdProtocolDescript
orList,*vProtocolDescriptor);

        CleanupStack::PopAndDestroy(vProtocolDescriptor);

        iSdpDatabase.UpdateAttributeL(recordAdvertiser,
        KSdpAttrIdBasePrimaryLanguage+KSdpAttrIdOffsetServiceName,
        _L("Serial Port"));

        iSdpDatabase.UpdateAttributeL(recordAdvertiser,
        KSdpAttrIdBasePrimaryLanguage+KSdpAttrIdOffsetServiceDescription
        ,_L("Simple point to point data transfer example"));
    }

    void BluetoothM::UpdateAvailability(bool b){
        TUint tmp;
        if (b)
           tmp = 0xFF;
        else
            tmp = 0x00;

        iSdpDatabase.UpdateAttributeL(recordAdvertiser,
        KSdpAttrIdServiceAvailability, tmp);
           iSdpDatabase.UpdateAttributeL(recordAdvertiser,
           KSdpAttrIdServiceRecordState, ++recordState);
    }
```

```
void BluetoothM::SetSecurityOnChannel(int channel){
    RBTMan secManager;

    RBTSecuritySettings secSettingsSession;

    User::LeaveIfError(secManager.Connect());
    CleanupClosePushL(secManager);
    User::LeaveIfError(secSettingsSession.Open(secManager));
    CleanupClosePushL(secSettingsSession);

    TBTServiceSecurity serviceSecurity(TUid::Uid(0x0FAA0067),
    KSolBtRFCOMM, 0);

    serviceSecurity.SetAuthentication(false);
    serviceSecurity.SetEncryption(false);
    serviceSecurity.SetAuthorisation(true);

    serviceSecurity.SetChannelID(channel);
    TRequestStatus status;
    secSettingsSession.RegisterService(serviceSecurity, status);

    User::WaitForRequest(status);
    User::LeaveIfError(status.Int());

    CleanupStack::PopAndDestroy();
    CleanupStack::PopAndDestroy();
}


BTSender::BTSender(BluetoothMObserver *bto,BluetoothM *_manager)
:CActive(EPriorityStandard),manager(_manager){
 parent = bto;
 CActiveScheduler::Add(this);
 //Inicialmente no esta enviando (este parametro es para saber si
 //esta ocupado enviando).
 sending = false;
}

void BTSender::RunL(){
 sending = false;

 if(iStatus != KErrNone){
 }else{
 //Aqui se nos informa de la confirmacion del envio
 }
}

void BTSender::CancelRequest(){
 sending = false;
 Cancel();
}

void BTSender::DoCancel(){
}

void BTSender::SendData(TBuf8<BT_PACKET_SIZE> &data){
 sending = true;
 socket->Write(data,iStatus);
 SetActive();
}
```

```
	void BTSender::SetSocket(RSocket *s){
	 socket = s;
	}

	bool BTSender::Busy(){
	 return sending;
	}

	BTReceiver::BTReceiver(BluetoothMObserver *bto,BluetoothM *_manager)
	:CActive(EPriorityStandard),manager(_manager){
	 parent = bto;
	 CActiveScheduler::Add(this) ;
	}

	void BTReceiver::RunL(){
	 if(iStatus != KErrNone){
	 }else{
		parent->BluetoothIncomingData(buffer);
		RequestData();
	 }
	}

	void BTReceiver::CancelRequest(){
	 Cancel();
	}

	void BTReceiver::DoCancel(){
	 socket->CancelRead();
	}

	void BTReceiver::RequestData(){
	 socket->RecvOneOrMore(buffer,0,iStatus,transferLength);
	 SetActive();
	}

	void BTReceiver::SetSocket(RSocket *s){
	 socket = s;
}
```

Una vez puestos los códigos se van a ver los aspectos más relevantes de los mismos. Solo decir que esta aplicación esta claro que no esta completa. Simplemente deberá acompañarse con un interfaz básico y añadir estas dos clases que son las que se encargan de todo.

No he puesto el código de la interfaz que cree yo para poder emularla y posteriormente probarla en los terminales porque me parece trivial.

Volviendo al código se puede observar que la clase encargada de manejar el BluetoothM hereda de BluetoothMObserver.

Para poner la aplicación a la escucha o realizar una conexión simplemente se debera llamar a los métodos Listen() o Connect() de la siguiente manera:

```
void BluetoothContainer::Listen(){
BluetoothM->Listen();
}

void BluetoothContainer::Connect(){
BluetoothM->Connect();
}
```

En este caso solo se intercambian datos cuando uno de los móviles pulsa una tecla ya que provoca un cambio en el sprite que el maneja. Es por eso que en el método de manejo de teclas esta el siguiente fragmento:

```
if(BluetoothM->state == eConnected){
//Si hay conexion mandar nueva posicion
TBuf8 data;
data.Append((unsigned char *)&x1,2);
data.Append((unsigned char *)&y1,2);
BluetoothM->Send(data,false);
}
```

En el BluetoothIncomingData cuando llegan datos de la posición del otro terminal se actualizan la X y la Y correspondientes para reflejar el cambio:

```
if(server){
Mem::Copy(&x2,data.Ptr(),2);
Mem::Copy(&y2,data.Ptr()+2,2);
}else{
if(!initReceived){
Mem::Copy(&x2,data.Ptr(),2);
Mem::Copy(&y2,data.Ptr()+2,2);
Mem::Copy(&x1,data.Ptr()+4,2);
Mem::Copy(&y1,data.Ptr()+6,2);
initReceived = true;
}else{
Mem::Copy(&x2,data.Ptr(),2);
Mem::Copy(&y2,data.Ptr()+2,2);
}
}
```

Es necesario hacer una distinción de si es la primera que se reciben datos, ya que si es la primera vez que el cliente recibe datos, no solo recibe la posición de la bola del servidor, si no que además el servidor le dice donde tiene que posicionar inicialmente la suya. Esto es para que una vez se inicie la conexión estén sincronizados.

Ahora se analizará un poco el método *BluetoothConnRequest*, que es donde el servidor recibe la confirmación de que ha recibido la conexión. Una vez sucede esto el servidor le manda las coordenadas de ambas bolas al cliente:

```
void BluetoothContainer::BluetoothConnRequest(){
//Se nos ha conectado alguien

//Para saber que somos el servidor de esta conexión
server = true;

TBuf8 data;

data.Append((unsigned char *)&x1,2);
```

```
data.Append((unsigned char *)&y1,2);
data.Append((unsigned char *)&x2,2);
data.Append((unsigned char *)&y2,2);
BluetoothM->Send(data,false);
}
```

Una vez vistas estas dos clases simplemente queda poner unas imágenes de la emulación del programa.

Figura 195. Captura 1 de pantalla de la aplicación bluetooth creada.

Figura 196. Captura 2 de pantalla de la aplicación bluetooth creada.

4.4.5 Desarrollo de aplicación final

4.4.5.1 Introducción

Como último apartado de la sección se va a desarrollar una aplicación utilizando todos los conceptos adquiridos a lo largo del capitulo. En este apartado ya no se verán mas librerías ni mas funciones, ya que con todo lo visto anteriormente se podrá realizar una aplicación de este tipo.

Por ello este capitulo esta basado por entero a la realización de una gran aplicación de tipo directorio. En ella se van a implementar todas las clases necesarias para crear una biblioteca de medios total para nuestro terminal UIQ Symbian. Esta aplicación se compondrá de varias clases que servirán para organizar todo lo que haya en el terminal. Se podrán agrupar todas las canciones en un espacio reservado llamado "Audio", todas las películas o videos en "Video", habrá también un apartado para libros llamado "Biblioteca", otro para juegos cuyo nombre será "Juegos" y una última categoría para el resto de archivos que no pertenezcan a ninguna de las anteriores la cual recibirá el nombre de "Sin Categoría". Habrá una sexta categoría llamada "Todos" donde se mostraran todos los tipos de archivos organizados en el directorio.

Figura 197. Captura de pantalla del entorno UIQ con el icono de la aplicación creada "Mi Directorio".

En esta aplicación directorio, podrán ser actualizadas por medio de menús, desplegables, textbox, etc. las informaciones concernientes a cada uno de los archivos contenidos en la misma, pudiendo ser modificados al antojo del usuario.

También se podrán seleccionar para ver, o para enviar a otros dispositivos. Se han añadido funcionalidades de Zoom por si se prefiere ver solo una parte de la pantalla (por ejemplo en la visualización de un video). Por supuesto podrán eliminarse los archivos que se quiera del directorio.

Además se podrá hacer una clasificación de cada tipo de archivo asignándole un número a elección de usuario para definir las preferencias (esto sobre todo sirve para el orden de la lista). De esta manera cada uno podrá ordenar sus listas como prefiera.

Como última nota mencionar que las categorías también son configurables, se podrán añadir las categorías que el usuario prefiera, asignándole el nombre que quiera a cada categoría.

4.4.5.2 Diagrama y Clases de la aplicación

Ahora se expondrá un diagrama con las clases que van a componer la aplicación que se pretende diseñar. Esta es una buena técnica de programación que permite ordenar ideas antes de lanzarse a la implementación de las mismas. Con ella se puede ver el concepto completo de la aplicación.

Es importante especificar un buen diagrama antes que otra cosa ya que facilitará mucho el trabajo de comprensión de la aplicación.

Con este diagrama se espera que quede clara la organización realizada para la aplicación antes de pasar a la implementación de las clases y funciones.

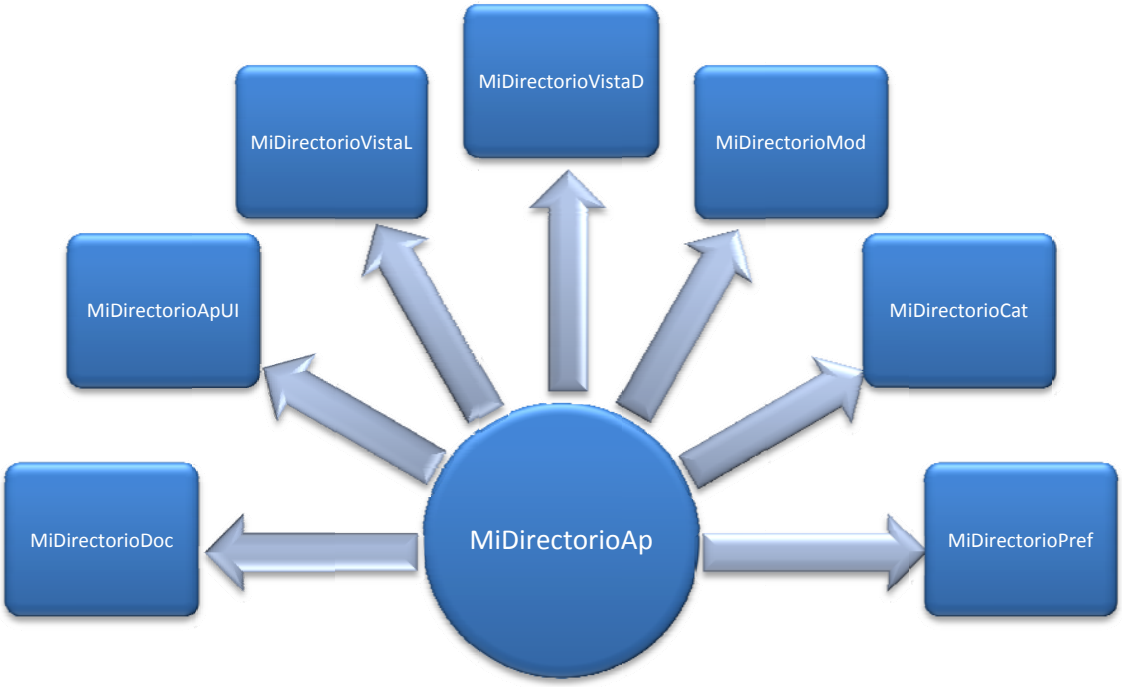


Figura 198. Esquema de la aplicación “Mi directorio ” con todas las clases que la componen.

MiDirectorioAp: Es el punto de entrada de la aplicación.

MiDirectorioDoc: Es responsable de almacenar todos los datos no volátiles en la memoria. También se encarga de definir los atributos del modelo y la aplicación.

MidirectorioApUI: Se encarga de crear la vista de la aplicación

MiDirectorioVistaL: Presenta todas las entradas de la aplicación Mi Directorio en una lista. Se pueden crear nuevas entradas y eliminar entradas antiguas.

MiDirectorioVistaD: Se usa para crear nuevos ítems en la aplicación y para mostrar los detalles de los ítems. Estos ítems pueden ser creados o eliminados. La vista de esta clase consiste en una página de tablas.

MiDirectorioMod: Esta clase es la encargada de controlar el almacenamiento de datos de la aplicación. Este modelo es usado para separar los datos de la aplicación del código de la UI. Se ha hecho así para facilitar los posibles cambios futuros en la UI.

MiDirectorioCat: Contiene todos los datos necesario para los ítems. Hace que sea posible almacenar los datos de forma no volátil.

MiDirectorioPref: Configura las preferencias que elija el usuario para la aplicación Mi Directorio.

4.4.5.3 Clases de la aplicación

En estas clases es donde se especifican las funciones que van a componer la aplicación. Son clases básicas, repletas de funciones implementadas y comentadas debidamente.

Se han obviado las clases de cabecera (.h) como en el resto de las aplicaciones, ya que son de sencilla construcción, simplemente son interfaces compuestas por espeficicaciones de las funciones sin implementar.

Estas clases de cabecera no se han puesto debido también a que suponen un engorro de espacio y código que no aporta nada a la comprensión y ejecución de las aplicaciones.

Con lo aprendido anteriormente simplemente viendo el código y con los comentarios que hay en el mismo es muy fácil comprender el funcionamiento de estas clases.

Clase MiDirectorioAp: Esta es la clase encargada del funcionamiento de la aplicación, ella se encarga de las funciones necesarias en cualquier programa realizado para Symbian.

```
#include <eikstart.h>

#include "MiDirectorioAp.h"
#include "MiDirectorioDoc.h"
#include "MiDirectorioIE.h" // Contiene el UID de la aplicación

TUid CMiDirectorioAp::AppDllUid() const
 {

 return KUidMiDirectorioApp;

 }

CApaDocument* CMiDirectorioAp::CreateDocumentL()
 {

 return CMiDirectorioDoc::NewL(*this);

 }

CApaApplication* NewApplication()
 {

 return new CMiDirectorioAp;

 }

GLDEF_C TInt E32Main()
{

 return EikStart::RunApplication(NewApplication);

}
```

Se compone de cuatro funciones las cuales se van a ver a continuación:

- La función *TUid CMiDirectorioAp::AppDllUid()* es llamada por el UI framework para preguntarle el UID de la aplicación. El valor devuelto es una constante, el UID exactamente.

- *CApaDocument* CMiDirectorioAp::CreateDocumentL(),* la utilidad de esta función es la de iniciar el framework y crear instancias del documento de la clase. Devuelve un puntero al documento .class creado.

- *CApaApplication* NewApplication(), e*sta función es llamada por el framework inmediatamente después de empezar el .exe de la aplicación. Esta función necesita crear un objeto aplicación o en caso de no poder crearlo devolverá null o fallo y devuelve una instancia de la clase aplicación.

- *GLDEF_C TInt E32Main(),* contiene el código de inicio del programa, el punto de entrada para el exe.

MiDirectorioApUI: Su función al igual que la de las demás esta explicada en el apartado anterior.

```
#include <QikEditCategoriesDlg.h>

#include "MiDirectorioApUI.h"
#include "MiDirectorioDoc.h"
#include "MiDirectorioListView.h"
#include "MiDirectorioVistaD.h"
#include "MiDirectorioIE.h"
#include "MiDirectorio.hrh"
#include <MiDirectorio.rsg>

void CMiDirectorioApUI::ConstructL()
 {
 // Llama a BaseConstructL para iniciar el UI..
 BaseConstructL();

 //Crea la vista de la lista y la añade al framework
 CMiDirectorioListView* MiDirectorioListView =
 CMiDirectorioListView::NewLC(*this, KNullViewId, iMiDirectorioMod);
 AddViewL(*MiDirectorioListView);
 CleanupStack::Pop(MiDirectorioListView);

 //Crea la vista de los detalles y la añade al framework
 TVwsViewId parentView(KUidMiDirectorioApp,
 KUidMiDirectorioListView);
 CMiDirectorioVistaD* MiDirectorioVistaD =
 CMiDirectorioVistaD::NewLC(*this, parentView, iMiDirectorioMod);
 AddViewL(*MiDirectorioVistaD);
 CleanupStack::Pop(MiDirectorioVistaD);

 // Crea la categoría del modelo y lo carga
 iCategoryModelInstance =
 QikCategoryUtils::ConstructCategoriesLC(R_MIDIRECTORIO_DEFAULT_CATE
 GORIES);
 CleanupStack::Pop(); // Categoria del modelo

 // Muestra los items
 iCurrentCategoryHandle = EMiDirectorioCatAll;
 }

/**Constructor para la aplicación.*/
CMiDirectorioApUI::CMiDirectorioApUI(CMiDirectorioMod& aModel) :
iMiDirectorioMod (aModel)
 {
 }

/**Destructor para la aplicación.*/
CMiDirectorioApUI::~CMiDirectorioApUI()
 {
 delete iCategoryModelInstance;
 }

/**Devuelve la categoría del modelo*/
CQikCategoryModel* CMiDirectorioApUI::CategoryModelInstance()
 {
 return iCategoryModelInstance;
 }
```

```
/**Devuelve la actual categoría del controlador*/
TInt CMiDirectorioApUI::CurrentCategoryHandle() const
 {
 return iCurrentCategoryHandle;
 }

/**Actualiza los parámetros del actual controlador de categorias*/
void CMiDirectorioApUI::SetCurrentCategoryHandle(TInt
aCategoryHandle)
 {

 if (iCategoryModelInstance->CategoryIndex(aCategoryHandle) !=
 KErrNotFound)
       {
       iCurrentCategoryHandle = aCategoryHandle;
       }
 else
       {
       iCurrentCategoryHandle = iCategoryModelInstance->
       HandleOfUnfiledCategory();
       }
 }

void CMiDirectorioApUI::HandleCategoryChangeL(TInt aCategoryHandle)
 {
 // Si la categria elegida ha cambiado actualiza el listbox.
 if (aCategoryHandle != CurrentCategoryHandle())
       {
       SetCurrentCategoryHandle(aCategoryHandle);

       CMiDirectorioListView* MiDirectorioListView =
       CurrentView<CMiDirectorioListView>();
       // Actualiza el listbox
       MiDirectorioListView->UpdateListBoxL();
       }
 }

/**Esta función es llamada cuando el usuario quiere editar una
categoría.*/
void CMiDirectorioApUI::EditCategoryL()
 {
 CQikEditCategoriesDialog::RunDlgLD(iCategoryModelInstance, this);
 }

TBool CMiDirectorioApUI::OkToAddCategory() const
 {
 return ETrue;
 }

TBool CMiDirectorioApUI::DoAddCategoryL(TInt& aHandle)
 {
 TQikCategoryName categoryName = iCategoryModelInstance->
 CategoryNameByHandle(aHandle);
 CategoryChangedL();
 return ETrue;
 }

/**Devuelve un parámetro que sirve para saber si una categoría puede
ser renombrada o no*/
TBool CMiDirectorioApUI::OkToRenameCategory(TInt , const TDesC& )
const
```

```
        {
        // Ningun error producido en el controlador es cogido por Qikon
        return ETrue;
        }

    /**Se llama cuando se renombra una categoría.Devuelve Etrue si se ha
    podido renombrar o Efalse en caso contrario.*/
    TBool CMiDirectorioApUI::DoRenameCategoryL(TInt aHandle, const
    TDesC& aNewName)
        {
        // Renombra la categoría en el modelo de la aplicacion
        iMiDirectorioMod.RenameCategory(aHandle, aNewName);
        CategoryChangedL();
        return ETrue;
        }

    /**Devuelve Etrue si una categoría puede ser mostrada en el momento
    actual, Efalse en caso contrario.*/
    TBool CMiDirectorioApUI::OkToMergeCategories(TInt , TInt) const
        {
        return EFalse;
        }

    TBool CMiDirectorioApUI::DoMergeCategoriesL(TInt , TInt )
        {
        return EFalse;
        }

    TBool CMiDirectorioApUI::OkToDeleteCategory(TInt ) const
        {
        return ETrue;
        }

    TBool CMiDirectorioApUI::DoDeleteCategoryL(TInt aHandle)
        {
        iMiDirectorioMod.RemoveCategory(aHandle);
        CategoryChangedL();

        // Comprueba si la categoría eliminada es la que actualmente se
        //muestra en pantalla.
        if(aHandle == iCurrentCategoryHandle)
            {
// Si es true, pone la actual categoría en Todas y actualiza el listbox
            iCurrentCategoryHandle = EMiDirectorioCatAll;
            CMiDirectorioListView* MiDirectorioListView =
CurrentView<CMiDirectorioListView>();
            // actualiza el listbox
            MiDirectorioListView->UpdateListBoxL();
            }
        return ETrue;
        }

/** Devuelve Etrue si una categoría concreta esta llena o EFalse si no.*/
    TBool CMiDirectorioApUI::IsCategoryEmpty(TInt /*aHandle*/) const
        {
        return ETrue;
        }

    void CMiDirectorioApUI::CategoryChangedL()
        {
```

```
 CMiDirectorioDoc* document = static_cast
<CMiDirectorioDoc*>(iDocument);
 document->SaveL();

 // Devuelve la vista de los detalles

 TVwsViewId viewId = TVwsViewId(KUidMiDirectorioApp,
 KUidMiDirectorioVistaD);

 CMiDirectorioVistaD* detailView =
(CMiDirectorioVistaD*)QikView(viewId);

 detailView->SetChoiceListDirty(ETrue);

}
```

Esta clase se compone de 18 funciones de las que se van a ver las más importantes:

- *void CMiDirectorioApUI::ConstructL()*, esta función crea la vista de la lista y los detalles y añade la vista al framework.

  Se devuelve el framework.

- *void CMiDirectorioApUI::HandleCategoryChangeL(TInt aCategoryHandle)*, esta función es llamada por el método que muestra las listas cuando recibe un comando de categoría.

  No devuelve nada pero llama al método de actualización del listbox.

- *TBool CMiDirectorioApUI::DoAddCategoryL(TInt& aHandle)*, sirve para añadir una nueva categoría. Añade dicha categoría al modelo de la aplicación y guarda el archivo.

- *TBool CMiDirectorioApUI::OkToDeleteCategory(TInt) const*, esta función sirve para saber si una categoría puede ser eliminada o no. Devuelve Etrue en caso afirmativo y Efalse en caso contrario.

  La siguiente función elimina la categoría en caso de que esta haya devuelto ETrue o no la elimina en caso de EFalse y devuelve Etrue si ha podido eliminarlo o EFalse en caso contrario.

- *void CMiDirectorioApUI::CategoryChangedL()*, salva el documento y configura la lista de elección.

Clase MiDirectorioCat: Esta es una clase muy sencilla de la que no hace falta comentar casi nada ni se destacará ninguna función ya que todas vienen comentadas y no entrañan ninguna dificultad. Solo decir que sirve para definir las categorías de los ítems y configurar los menús desplegables y los rankings.

```
#include "MiDirectorioCat.h"

/**Devuelve el controlador de la categoria*/
TInt TMiDirectorioCat::CategoryHandle() const
 {
 return iCategoryHandle;
 }

/**Devuelve el nombre de la categoria*/
TQikCategoryName TMiDirectorioCat::CategoryName() const
 {
 return iCategoryName;
 }

/**Configura el controlador de la categoría.*/
void TMiDirectorioCat::SetCategoryHandle(TInt aCategoryHandle)
 {
 iCategoryHandle = aCategoryHandle;
 }

/**Configura el nombre de la categoria*/
void TMiDirectorioCat::SetCategoryName(TQikCategoryName
aCategoryName)
 {
 iCategoryName = aCategoryName;
 }

/** Guarda los datos en un stream.*/
void TMiDirectorioCat::ExternalizeL(RWriteStream& aStream) const
 {

 aStream.WriteInt32L(iCategoryHandle);

 // Escribe los nombre en un stream
 aStream << iCategoryName;
 }

/** Restaura los datos procedentes del Stream*/
void TMiDirectorioCat::InternalizeL(RReadStream& aStream)
 {

 // Crea un buffer temporal para guardar el nombre de la categoria
 TBuf<EQikCategoryNameMaxLength> categoryName;

 iCategoryHandle = aStream.ReadInt32L();

 // Lee el nombre desde el stream
 aStream >> categoryName;
 iCategoryName = categoryName;
}
```

Clase MiDirectorioVistaD: Clase encargada de aguantar toda la interfaz gráfica de la aplicación, esta clase lo dibuja todo, y se encarga de actualizar las pantallas en caso necesario. Esta compuesta por una gran cantidad de funciones de dibujo de las cuales al final del código se resaltarán las más importantes.

```
#include <QikCommand.h> // CQikCommand
#include <QikNumberEditor.h> // CQikNumberEditor
#include <eikchlst.h> // CEikChoiceList
#include <QikZoomDlg.h> // CQikZoomDialog
#include <eikrted.h> // CEikEdwin & CEikRichTextEditor
#include <eiklabel.h> // CEikLabel
#include <QikSaveChangesDlg.h> // SaveChangesDialog

#include "MiDirectorioVistaD.h"
#include "MiDirectorioApUI.h"
#include "MiDirectorioDoc.h"
#include "MiDirectorioMod.h"
#include "MiDirectorioIt.h"
#include "MiDirectorioIE.h"
#include "MiDirectorio.hrh"
#include "MiDirectorioCat.h"
#include <MiDirectorio.rsg>

/**Inicializo el ranking con un valor default de “2”. */
const TInt KDefaultRank = 2;

/**Constructor de la vista de los detalles*/
CMiDirectorioVistaD* CMiDirectorioVistaD::NewLC(CQikAppUi& aAppUi,
const TVwsViewId aParentViewId, CMiDirectorioMod& aModel)
 {
 CMiDirectorioVistaD* self = new (ELeave)
 CMiDirectorioVistaD(aAppUi, aParentViewId, aModel);
 CleanupStack::PushL(self);
 self->ConstructL();
 return self;
 }

/**Constructor para la vista. Añade las referencias al modelo de a
aplicación y al manejador de comandos.*/
CMiDirectorioVistaD::CMiDirectorioVistaD(CQikAppUi& aAppUi, const
TVwsViewId aParentViewId, CMiDirectorioMod& aModel)
 : CQikMultiPageViewBase(aAppUi, aParentViewId),
iMiDirectorioMod(aModel),
iCommandManager(CQikCommandManager::Static(*iCoeEnv))
 {
 }

/**Destructor para la vista de los detalles*/
CMiDirectorioVistaD::~CMiDirectorioVistaD()
 {
 delete iDefaultName;
 delete iCategoryUnfiled;

 if(iCurrentItem)
       {
       const TBool iCurrentItemOwnedByModel = (KNullUid !=
       iCurrentItem->Uid());
       if(!iCurrentItemOwnedByModel)
             {
```

```
            delete iCurrentItem;
            }
      }
 iCategoryChoiceListMatch.Close();
 }

/**Lee el nombre de la vista de los detalles*/
void CMiDirectorioVistaD::ConstructL()
 {
 // Llama a ConstructL para iniciarlo con valores estandar.
 BaseConstructL();

 //Lee el nombre estandar.
 iDefaultName = iEikonEnv->
 AllocReadResourceL(R_MIDIRECTORIO_DETAILVIEW_DEFAULT_NAME_TEXT);
 iCategoryUnfiled = iEikonEnv->
 AllocReadResourceL(R_MIDIRECTORIO_DETAILVIEW_CATEGORY_UNFILED);

 iChoiceListDirty = ETrue;
 }

/**Crea la vista del framework e inicializa el valor del zoom.*/
void CMiDirectorioVistaD::ViewConstructL()
 {
 // Carga la información acerca de la configuración del UIs
 ViewConstructFromResourceL(R_MIDIRECTORIO_DETAILVIEW_UI_CONFIGURATI
 ONS, R_MIDIRECTORIO_DETAILVIEW_CONTROLS);

 //Devuelve el estado del zoom
 CMiDirectorioDoc* document = static_cast <CMiDirectorioDoc*>
 (iQikAppUi.Document());
 SetZoomFactorL(document->Preferences().DetailViewZoomState());

 //Actualiza la choice list
 UpdateChoiceListL();

 //Inicializa los controles a sus valores default.
 InitControlsL();
 }

/**Devuelve la vista de los detalles*/
TVwsViewId CMiDirectorioVistaD::ViewId()const
 {
 return TVwsViewId(KUidMiDirectorioApp, KUidMiDirectorioVistaD);
 }

void CMiDirectorioVistaD::HandleCommandL(CQikCommand& aCommand)
 {
 switch(aCommand.Id())
      {
      case EMiDirectorioViewDSaveCmd:
            {
            SaveItemL();
            break;
            }

      case EMiDirectorioViewDDeleteCmd:
            {
            //Si es un ítem nuevo no es necesario eliminarlo
            if(KNullUid != iCurrentItem->Uid())
                  {
```

```
                if( iEikonEnv-> QueryWinL
                (R_MIDIRECTORIO_DIALOG_DELETE_UNDOABLE_TITLE,
                R_MIDIRECTORIO_DIALOG_DELETE_UNDOABLE_LABEL_TEXT)
                )
                        {
                        DeleteItemL();
                        }
                }
        break;
        }
case EMiDirectorioViewDZoomCmd:
        {
        // Lanza el menú del zoom
        CMiDirectorioDoc* document = static_cast
        <CMiDirectorioDoc*>(iQikAppUi.Document());

        const TInt zoomFactor =
        CQikZoomDialog::RunDlgLD(document->
        Preferences().DetailViewZoomState());

        // Si el estado del zoom cambia salta un evento
        if(document->
        Preferences().SetDetailViewZoomState(zoomFactor))
                {
                // Define el factor del zoom
                SetZoomFactorL(zoomFactor);
                }
        break;
        }
case EQikCmdGoBack:
        {
        TInt exitOption = CQikSaveChangesDialog::ECancel;
        if(iDataDirty)
                {
                // Lanza el menú para salvar los cambios
                exitOption = CQikSaveChangesDialog::RunDlgLD();

                if(exitOption == CQikSaveChangesDialog::ESave)
                        {
                        // Salva el ítem y devuleve la lista
                        SaveItemL();
                        break;
                        }
                }

        if(!iDataDirty || exitOption !=
        CQikSaveChangesDialog::ESave)
                {

                if(iCurrentItem)
                        {
                        TBool iCurrentItemOwnedByModel = (KNullUid
                        != iCurrentItem->Uid());

                        if(!iCurrentItemOwnedByModel)
                                {
                                delete iCurrentItem;
                                }
                        }
```

```
                    //Resetea el foco
                    RequestFocusL(NULL);
                    }

              //Inicia todos los controles a los valores default
              InitControlsL();
              }
        default:
              CQikMultiPageViewBase::HandleCommandL(aCommand);
              break;
        }
  }

void CMiDirectorioVistaD::HandleControlEventL(CCoeControl *aControl,
TCoeEvent aEventType)
  {
  if(aEventType == EEventStateChanged)
        {
        switch(aControl->UniqueHandle())
              {
              case EMiDirectorioVistaDNameCtrl:
              case EMiDirectorioVistaDYearCtrl:
              case EMiDirectorioVistaDRankCtrl:
              case EMiDirectorioVistaDCategoryCtrl:
              case EMiDirectorioVistaDNoteCtrl:
                    {
                    iDataDirty = ETrue;
                    }
              }
        }
  CQikMultiPageViewBase::HandleControlEventL(aControl, aEventType);
  }

void CMiDirectorioVistaD::TabActivatedL(TInt aTabId)
  {
  if(aTabId == EMiDirectorioVistaDNotePage)
        {
        // Busca la etiqueta para el control de captura
        CEikLabel* labelCtrl = LocateControlByUniqueHandle
        <CEikLabel>(EMiDirectorioVistaDNoteCaptionCtrl);

        // Busca el nombre del control de captura
        CEikEdwin* edwinCtrl = LocateControlByUniqueHandle
        <CEikEdwin>(EMiDirectorioVistaDNameCtrl);

        if(edwinCtrl->Text()->DocumentLength() > 0)
              {
              TBuf<EMiDirectorioNameMaxLength> nameText(KNullDesC());
              edwinCtrl->GetText(nameText);
              labelCtrl->SetTextL(nameText);
              }
        else
              {
              // Usa el texto default para la etiquetal
              labelCtrl->SetTextL(*iDefaultName);
              }
        labelCtrl->DrawDeferred();
        }
  CQikMultiPageViewBase::TabActivatedL(aTabId);
  }
```

```
void CMiDirectorioVistaD::ViewActivatedL(const TVwsViewId&
aPrevViewId, const TUid aCustomMessageId, const TDesC8& aCustomMessage)
	{
	// Actualiza el choice listt
	if(iChoiceListDirty)
		{
		UpdateChoiceListL();
		}

	if(ActivePageId() != EMiDirectorioVistaDDetailPage)
		{
		ActivatePageL(EMiDirectorioVistaDDetailPage);
		}

	if(aCustomMessageId == KUidMiDirectorioDnlItem)
		{
		// Actualiza un ítem
		TMiDirectorioDnlItemBuf buf;
		buf.Copy(aCustomMessage);

		if(buf().iState == EUpdateItem)
			{
			// Devuelve un ítem al modelo
			TUid currentUid = buf().iUid;
			iCurrentItem = iMiDirectorioMod.Item(currentUid);

			UpdateViewFromItemL();
			}
		}
	else if(aPrevViewId.iAppUid == KUidMiDirectorioApp)
		{
		// Crea un ítem nuevo
		iCurrentItem = CMiDirectorioIt::NewL();
		}

	// Decide si el comando eliminar se puede utilizar en este momento
	TBool iExistingItem = (KNullUid != iCurrentItem->Uid());
	iCommandManager.SetAvailable(*this, EMiDirectorioVistaDDeleteCmd,
	iExistingItem);
	}

/**Cuando un comando inserta una lista de comandos nueva, el cliente
recibe una llamada para iniciar el comando. Esta función se encarga
de realizar esto y de cambiar el texto.*/
CQikCommand* CMiDirectorioVistaD ::
DynInitOrDeleteCommandL(CQikCommand* aCommand, const CCoeControl& {
	switch(aCommand->Id())
		{
		case EQikCmdGoBack:
			{
			aCommand-> SetTextL
			(R_MIDIRECTORIO_DETAILVIEW_CANCEL_COMMAND_TEXT);
			aCommand->SetIcon(static_cast<const CQikContent*>
			(NULL), ETakeOwnership);
			break;
			}
		default:
			break;
		}
	return aCommand;
	}
```

```
/**Inicia todos los controles a sus valores default*/
void CMiDirectorioVistaD::InitControlsL()
 {

 CEikEdwin* edwinCtrl = LocateControlByUniqueHandle <CEikEdwin>
 (EMiDirectorioVistaDNameCtrl);
 edwinCtrl->SetTextL(NULL);

 //Cambia el año en un detaller.
 CQikNumberEditor* numberEditorCtrl = LocateControlByUniqueHandle
 <CQikNumberEditor>(EMiDirectorioVistaDYearCtrl);

 // Tiempo en microsegundos
 TTime time;

 //Define y pinta la hora y la fecha
 time.UniversalTime();
 TDateTime dateTime(time.DateTime());

 // Define el año actualr
 numberEditorCtrl->SetValueL(dateTime.Year());

 // Cambia el ranking de la choicelist.
 CEikChoiceList* choiceListCtrl = LocateControlByUniqueHandle
 <CEikChoiceList>(EMiDirectorioVistaDRankCtrl);
 choiceListCtrl->SetCurrentItem(KDefaultRank);

 //Cambia la categoría de la choicelist.
 choiceListCtrl = LocateControlByUniqueHandle
 <CEikChoiceList>(EMiDirectorioVistaDCategoryCtrl);
 choiceListCtrl-> SetCurrentItem
(ChoiceIndexByHandle(EMiDirectorioCatUnfiled));

 // Cambia la etiqueta del texto.
 CEikLabel* labelCtrl = LocateControlByUniqueHandle <CEikLabel>
 (EMiDirectorioVistaDNoteCaptionCtrl);
 labelCtrl->SetTextL(*iDefaultName);

 CEikRichTextEditor* richTextEditorCtrl LocateControlByUniqueHandle
 <CEikRichTextEditor>(EMiDirectorioVistaDNoteCtrl);
 richTextEditorCtrl->SetTextL(NULL);

 // Reinicia el flag del tiempo.
 iDataDirty = EFalse;
 }

/**Es  llamado cuando se abre un ítem. Devuelve los valores del
item.*/
void CMiDirectorioVistaD::UpdateViewFromItemL()
 {
 if(iCurrentItem)
       {

       CEikEdwin* edwinCtrl = LocateControlByUniqueHandle
       <CEikEdwin>(EMiDirectorioVistaDNameCtrl);
       edwinCtrl->SetTextL(&iCurrentItem->Name());
```

```
 // Cambia el año en el editor de los detalles
       CQikNumberEditor* numberEditorCtrl =
       LocateControlByUniqueHandle<CQikNumberEditor>(EMiDirectorioVi
       staDYearCtrl);
       numberEditorCtrl->SetValueL(iCurrentItem->Year());


       CEikChoiceList* choiceListCtrl = LocateControlByUniqueHandle
       <CEikChoiceList>(EMiDirectorioVistaDRankCtrl);
       choiceListCtrl->SetCurrentItem(iCurrentItem->Rank());


       choiceListCtrl = LocateControlByUniqueHandle
       <CEikChoiceList>(EMiDirectorioVistaDCategoryCtrl);
       choiceListCtrl->SetCurrentItem
       (ChoiceIndexByHandle(iCurrentItem->Category()));


       CEikRichTextEditor* richTextEditorCtrl =
       LocateControlByUniqueHandle<CEikRichTextEditor>(EMiDirectorio
       VistaDNoteCtrl);
       richTextEditorCtrl->SetTextL(&iCurrentItem->Note());
       }
 }

/**Salva los valores actuales del controlador del ítem abierto*/
void CMiDirectorioVistaD::UpdateItemFromViewL()
 {
 if(iCurrentItem)
       {
       // Define el nombre del item.
       CEikEdwin* edwinCtrl = LocateControlByUniqueHandle
       <CEikEdwin>(EMiDirectorioVistaDNameCtrl);
       // Comprueba si existe otro ítem con el mismo nombre
       if(edwinCtrl->Text()->DocumentLength() > 0)
             {
             TBuf<EMiDirectorioNameMaxLength> nameText(KNullDesC());
             edwinCtrl->GetText(nameText);
             iCurrentItem->SetNameL(nameText);
             }
       else
             {
             iCurrentItem->SetNameL(*iDefaultName);
             }

       // Define la fecha en los detalles del item.
       CQikNumberEditor* numberEditorCtrl =
       LocateControlByUniqueHandle<CQikNumberEditor>(EMiDirectorioVi
       staDYearCtrl);
       iCurrentItem->SetYear(numberEditorCtrl->Value());

       // Define el ranking en los detalles del item.
       CEikChoiceList* choiceListCtrl =
       LocateControlByUniqueHandle<CEikChoiceList>(EMiDirectorioVist
       aDRankCtrl);
       iCurrentItem->SetRank(choiceListCtrl->CurrentItem());

       // Define la categoría en los detalles del item.
       choiceListCtrl = LocateControlByUniqueHandle
       <CEikChoiceList>(EMiDirectorioVistaDCategoryCtrl);
```

```
        TInt handle = iCategoryChoiceListMatch[choiceListCtrl->
        CurrentItem()];
        iCurrentItem->SetCategory(handle);

        // Define la notas de los items.
        CEikRichTextEditor* richTextEditorCtrl =
        LocateControlByUniqueHandle<CEikRichTextEditor>(EMiDirectorio
        VistaDNoteCtrl);

        HBufC* noteText = richTextEditorCtrl->GetTextInHBufL();
        CleanupStack::PushL(noteText);
        if(noteText)
              {
              iCurrentItem->SetNoteL(*noteText);
              }
        CleanupStack::PopAndDestroy(noteText);

        // Avisa si el modelo a cambiado
        iMiDirectorioMod.SetDirty(ETrue);
        }
 }

void CMiDirectorioVistaD::SaveL()
 {
 CMiDirectorioDoc* document = static_cast <CMiDirectorioDoc*>
 (iQikAppUi.Document());
 // si es necesario guardar los datos de la aplicación en la memoria
 //no volátil salta
 if(document->Preferences().IsDirty() || iMiDirectorioMod.IsDirty())
        {
        document->SaveL();
        }
 }

void CMiDirectorioVistaD::SaveItemL()
 {
 //Guarda los valores actualres del controlador del item
 UpdateItemFromViewL();

 const TBool newItem = (KNullUid == iCurrentItem->Uid());
 if(newItem)
        {
        iMiDirectorioMod.AddItemL(iCurrentItem);
        }

 // Crea un mensaje para recordar la lista
 TMiDirectorioDnlItem entry;
 entry.iState = newItem ? EAddItem : EUpdateItem;
 entry.iUid = iCurrentItem->Uid();
 TMiDirectorioDnlItemBuf message(entry);

 TVwsViewId listView = TVwsViewId(KUidMiDirectorioApp,
 KUidMiDirectorioListView);
 iQikAppUi.ActivateViewL(listView, KUidMiDirectorioDnlItem,
 message);

 // Inicializa los controles a su valor default.
 InitControlsL();

 RequestFocusL(NULL);
 }
```

```
/**Elimina un ítem del modelo de la aplicación.*/
void CMiDirectorioVistaD::DeleteItemL()
 {

 TMiDirectorioDnlItem dnl;
 dnl.iState = EDeleteItem;
 dnl.iUid = iCurrentItem->Uid();
 TMiDirectorioDnlItemBuf message(dnl);

 // Elimina el iteml
 iMiDirectorioMod.RemoveItem(iCurrentItem->Uid());

 TVwsViewId listView = TVwsViewId(KUidMiDirectorioApp,
 KUidMiDirectorioListView);
 iQikAppUi.ActivateViewL(listView, KUidMiDirectorioDnlItem,
 message);


 InitControlsL();

 RequestFocusL(NULL);
 }

/**Define el factor del zoom
void CMiDirectorioVistaD::SetZoomFactorL(TInt aZoomLevel)
 {
 const TInt zoomFactor = CQikAppUi::ZoomFactorL(aZoomLevel,
 *iEikonEnv);
 CQikMultiPageViewBase::SetZoomFactorL(zoomFactor);
 }

void CMiDirectorioVistaD::UpdateChoiceListL()
 {
 CEikChoiceList* choiceListCtrl = LocateControlByUniqueHandle
 <CEikChoiceList>(EMiDirectorioVistaDCategoryCtrl);
 iCategoryChoiceListMatch.Reset();

 // lee las categorías de la memoriae
 CDesCArray* resourceArray = iCoeEnv-> ReadDesCArrayResourceL
 (R_MIDIRECTORIO_DETAILVIEW_CATEGORY_ITEMS);
 CleanupStack::PushL(resourceArray);

 // Añade categorías adicionales nuevas
 TInt count = iMiDirectorioMod.CategoryCount();
 for(TInt index = 0; index < count; index++)
       {
       TMiDirectorioCat& category =
       iMiDirectorioMod.Category(index);
       resourceArray->AppendL(category.CategoryName());
       }

 // Ordena el array alfabeticamente
 resourceArray->Sort(ECmpFolded);

 //Añade los archivos sin un nombre espcifico al final del array
 resourceArray->AppendL(*iCategoryUnfiled);

 choiceListCtrl->SetArrayL(resourceArray);
 CleanupStack::Pop(resourceArray);
```

```
 CDesC16Array& choiceArray = *(choiceListCtrl->DesCArray());
 count = choiceArray.Count();
 CQikCategoryModel* categoryModel = static_cast <CMiDirectorioApUI*>
 (iCoeEnv->AppUi())->CategoryModelInstance();
 for(TInt index = 0; index < count; index++)
       {
       TPtrC16 choice = choiceArray[index];
       TInt handle = categoryModel->CategoryHandle(choice);
       iCategoryChoiceListMatch.Append(handle);
       }

 iChoiceListDirty = EFalse;
 }

/**Busca en el array una coincidencia con el parámetro que se le
pasa a la función y devuelve el id de su controlador*/
TInt CMiDirectorioVistaD::ChoiceIndexByHandle(TInt aHandle)
 {
 TInt count = iCategoryChoiceListMatch.Count();
 for(TInt index = 0; index < count; index++)
       {
       TInt value = iCategoryChoiceListMatch[index];
       if(value == aHandle)
             {
             return index;
             }
       }
 return KErrNotFound;
 }

/**Devuelve la choice list*/
TInt CMiDirectorioVistaD::IsChoiceListDirty() const
 {
 return iChoiceListDirty;
 }

void CMiDirectorioVistaD::SetChoiceListDirty(TBool aDirty)
{
 iChoiceListDirty = aDirty;
}
```

En esta clase solo se va a hablar más en profundidad de las funciones que manejan los eventos, ya que es de las cosas más nuevas que se han visto y quizá no estén totalmente claras:

- *void CMiDirectorioVistaD::HandleCommandL(CQikCommand& aCommand)*, maneja todos los comandos de la vista. Llama al framework cuando un comando salta. Esta función es un claro ejemplo de cómo se manejan todo tipo de eventos.

- *void CMiDirectorioVistaD::HandleControlEventL(CCoeControl *aControl, TCoeEvent aEventType)*, se encarga de manejar los eventos generados por el controlador. También controla que no se produzcan cambios en los datos.

- *void CMiDirectorioVistaD::ViewActivatedL(const TVwsViewId& aPrevViewId, const TUid aCustomMessageId, const TDesC8& aCustomMessage),* cuando la aplicación necesita mostrar una vista se llama a esta función. La cual pasa la vista como parámetro a la función que lo pide. También se usa para actualizar o eliminar ítems de la vista de detalles.

- *void CMiDirectorioVistaD::SaveL(),* esta función llama al .class del documento para guardar el modelo de la aplicación. Y es llamada después de que el reloj del modelo de la aplicación haya cambiado.

- *void CMiDirectorioVistaD::UpdateChoiceListL(),* función utilizada para actualizar la choice list de las categorías. Para ello lee del archivo de reconocimiento y de la memoria del modelo de la aplicación. Añade todas las categorías a un array, las ordena y las envía a la choice list.

Clase MiDirectorioDoc: Esta clase sirve para guardar todo el directorio, todos los ítems, etc. Ella se encargará de que los datos sean guardados en una memoria no volátil para no perderlos. Al final del código se comentan las funciones más importantes del mismo.

```
#include <s32strm.h>
#include <QikApplication.h>

#include "MiDirectorioDoc.h"
#include "MiDirectorioApUI.h"
#include "MiDirectorioMod.h"
#include "MiDirectorioIE.h"
#include "MiDirectorioPreferences.h"

/**Constructor principal de la clase.*/
CMiDirectorioDoc* CMiDirectorioDoc::NewL(CQikApplication& aApp)
 {
 CMiDirectorioDoc* self = new (ELeave) CMiDirectorioDoc(aApp);
 CleanupStack::PushL(self);
 self->ConstructL();
 CleanupStack::Pop(self);
 return self;
 }

/**Constructor de la clase*/
CMiDirectorioDoc::CMiDirectorioDoc(CQikApplication& aApp)
 : CQikDocument(aApp)
 {
 }

void CMiDirectorioDoc::ConstructL()
 {
 // Crea el modelo que guardará todos los items
 iMiDirectorioMod = CMiDirectorioMod::NewL();
 }

/**Función que elimina el modelo del directorio cuando se termina de
utilizar*/
CMiDirectorioDoc::~CMiDirectorioDoc()
```

```
 {
 delete iMiDirectorioMod;
 }

/**Devuelve una referencia al modelo*/
CMiDirectorioMod& CMiDirectorioDoc::MiDirectorioMod()
 {
 return *iMiDirectorioMod;
 }

/**Devuelve los atributos*/
TMiDirectorioPreferences& CMiDirectorioDoc::Preferences()
 {
 return iPreferences;
 }

void CMiDirectorioDoc::StoreL(CStreamStore& aStore,
CStreamDictionary& aStreamDic) const
 {
 RStoreWriteStream stream;

 TStreamId preferenceId = stream.CreateLC(aStore);
 aStreamDic.AssignL(KUidMiDirectorioStore, preferenceId);
 stream << iPreferences;
 iMiDirectorioMod->StoreL(stream);

 // sirve para asegurar que ningún buffer esta escribiendo en el
 //disco
 stream.CommitL();
 CleanupStack::PopAndDestroy(); // stream
 }

/**Llama al framework para iniciar la aplicación y guarda los datos
en el disco.*/
void CMiDirectorioDoc::RestoreL(const CStreamStore& aStore, const
CStreamDictionary& aStreamDic)
 {
 // Busca el ID del stream
 TStreamId preferenceId(aStreamDic.At(KUidMiDirectorioStore));
 RStoreReadStream stream;
 stream.OpenLC(aStore, preferenceId);
 if(preferenceId != KNullStreamId)
       {
       stream >> iPreferences;
       iMiDirectorioMod->RestoreL(stream);
       }

 CleanupStack::PopAndDestroy(); // stream
 }


CEikAppUi* CMiDirectorioDoc::CreateAppUiL()
 {
 return new (ELeave) CMiDirectorioApUI(*iMiDirectorioMod);
 }
```

Las funciones más importantes de esta clase son:

- *void CMiDirectorioDoc::StoreL(CStreamStore& aStore, CStreamDictionary& aStreamDic) const*, esta función es llamada por el framework cuando necesita guardar el factor del zoom.

- *CEikAppUi* CMiDirectorioDoc::CreateAppUiL()*, esta función sirve para crear una instancia del UI de la aplicación cuando el framework la llama.

Clase MiDirectorioMod: Con esta clase se crea el modelo de la aplicación, usando el paradigma anteriormente visto MVC (modelo-vista-controlador). Al final del código se han comentado las funciones más importantes de la misma.

```
#include <s32strm.h>

#include "MiDirectorioMod.h"
#include "MiDirectorioIt.h"
#include "MiDirectorioCat.h"
#include "MiDirectorio.hrh"

/**Constructor*/
CMiDirectorioMod* CMiDirectorioMod::NewL()
 {
 CMiDirectorioMod* self = new (ELeave) CMiDirectorioMod();
 CleanupStack::PushL(self);
 self->ConstructL();
 CleanupStack::Pop(self);
 return self;
 }

/**Constructor secundario para el modelo*/
CMiDirectorioMod::CMiDirectorioMod()
 {
 }

void CMiDirectorioMod::ConstructL()
 {
 }

/**Destructor*/
CMiDirectorioMod::~CMiDirectorioMod()
 {
 iMiDirectorioIts.ResetAndDestroy();
 iMiDirectorioIts.Close();
 iMiDirectorioCategories.Close();

 delete iDeletedItem;
 }

/**Esta función se llama cuando se añade un nuevo ítem al modelo*/
void CMiDirectorioMod::AddItemL(CMiDirectorioIt* aItem)
 {
 ASSERT(aItem);

 aItem->SetUid(NewUniqueUid());
 iMiDirectorioIts.AppendL(aItem);
 iDirty = ETrue;
```

```
 }

/**Esta función es llamada cuando se elimina un ítem del modelo.*/
void CMiDirectorioMod::RemoveItem(TInt aIndex)
 {
 ASSERT(aIndex > KErrNotFound);

 if (aIndex > KErrNotFound)
       {
       if(iDeletedItem)
             {
             delete iDeletedItem;
             iDeletedItem = NULL;
             }

       iDeletedItem = Item(aIndex);

       // Elimina el ítem del array
       iMiDirectorioIts.Remove(aIndex);

       iDirty = ETrue;
       }
 }

/**Esta función se llama cuando se quiere eliminar un ítem, para
confirmar que todavía pertence al array*/
void CMiDirectorioMod::RemoveItem(const TUid& aUid)
 {

 // Busca la posición del ítem dentro del directorio
 const TInt index = FindItemIndexFromMiDirectorioUid(aUid);
 RemoveItem(index);
 }

/**Esta función se llama cuando se quieren eliminar varios ítems a
la vez.*/
void CMiDirectorioMod::RemoveItem(RArray<TUid> aUids)
 {
 const TInt count = aUids.Count();

 if(count == 1)
       {
       // Si solo existe un ítem en el array te ofrece la
       //posibilidad de volver atrás sin eliminarlo
       RemoveItem(aUids[0]);
       }
 else
       {
       CMiDirectorioIt* item = NULL;


       if(iDeletedItem)
             {
             delete iDeletedItem;
             iDeletedItem = NULL;
             }

       // Bucle para buscar el id del ítem a eliminar
       for(TInt i = 0; i < count; i++)
             {
```

```
                // Busca la posición del ítem dentro del array
                const TInt index =
                FindItemIndexFromMiDirectorioUid(aUids[i]);
                if(index > KErrNotFound)
                        {
                        // Elimina el ítem del array.
                        item = Item(index);
                        iMiDirectorioIts.Remove(index);
                        delete item;
                        item = NULL;
                        }
                }
        iDirty = ETrue;
        }
 }

/**Función para recuperar un ítem anteriormente eliminado*/
void CMiDirectorioMod::RecoverDeletedItemL()
 {

 if(iDeletedItem)
        {

        AddItemL(iDeletedItem);
        iDeletedItem = NULL;
        iDirty = ETrue;
        }
 }

TBool CMiDirectorioMod::DeletedItemExist() const
 {
 return iDeletedItem != NULL;
 }

CMiDirectorioIt* CMiDirectorioMod::Item(const TInt aIndex)
 {

 return iMiDirectorioIts[aIndex];
 }

CMiDirectorioIt* CMiDirectorioMod::Item(const TUid&
 aMiDirectorioItId)
 {
 // Finds the index position for the my directory item
 const TInt index =
 FindItemIndexFromMiDirectorioUid(aMiDirectorioItId);
 if(index > KErrNotFound)
        {

        return iMiDirectorioIts[index];
        }
 else
        {
        return NULL;
        }
 }

/**Devuelve el numero de ítems exitentes*/
TInt CMiDirectorioMod::ItemCount() const
 {
```

```
return iMiDirectorioIts.Count();
 }

/**Llama al función que se utiliza para buscar los items*/
TInt CMiDirectorioMod::FindItemIndexFromMiDirectorioUid(const TUid&
aUid) const
 {
 const TInt count = iMiDirectorioIts.Count();
 for(TInt index = 0; index < count; index++)
       {
       if(iMiDirectorioIts[index]->Uid() == aUid)
             {
             return index;
             }
       }
 return KErrNotFound;
 }

/**Guarda en un strem el directorio con todos los ítems.*/
void CMiDirectorioMod::StoreL(RWriteStream& aStream) const
 {

 const TInt count = iMiDirectorioIts.Count();

 // Escribe el numero de ítems que existen en el array
 aStream.WriteInt32L(count);

 // Lo escribe en el stream
 for(TInt index = 0; index < count; index++)
       iMiDirectorioIts[index]->ExternalizeL(aStream);

 const TInt categoryCount = iMiDirectorioCategories.Count();
 aStream.WriteInt32L(categoryCount);

 // Escribe la categoría en el stream
 for(TInt index = 0; index < categoryCount; index++)
       {
       iMiDirectorioCategories[index].ExternalizeL(aStream);
       }

 // Le dice al modelo que ya todo a sido guardado
 const_cast<CMiDirectorioMod*>(this)->iDirty = EFalse;
 }

/**Carga los items*/
void CMiDirectorioMod::RestoreL(RReadStream& aStream)
 {

 TInt count = aStream.ReadInt32L();
 while(count--)
       {

       // Crea y lee los ítems
       CMiDirectorioIt* item = CMiDirectorioIt::NewLC();
       item->InternalizeL(aStream);
       AddItemL(item);
       CleanupStack::Pop(item);
       }
```

```
 count = aStream.ReadInt32L();
 if(count > 0)
       {
       while(count--)
             {
             // Lee y crea la categoría
             TMiDirectorioCat category;
             category.InternalizeL(aStream);

             // añade la categoría al array
             AddCategory(category);
             }
       }
 }

/**Salta si el estado de los ítems ha sido cambiado*/
TBool CMiDirectorioMod::IsDirty() const
 {
 return iDirty;
 }

void CMiDirectorioMod::SetDirty(TBool aDirty)
 {
 iDirty = aDirty;
 }

/**Añade la categoría al array de categorias*/
void CMiDirectorioMod::AddCategory(const TQikCategoryName&
aCategoryName, TInt& aHandle)
 {

 TMiDirectorioCat category;
 category.SetCategoryHandle(aHandle);
 category.SetCategoryName(aCategoryName);
 iMiDirectorioCategories.Append(category);
 iDirty = ETrue;
 }

/**Igual que el anterior pero para otra categoria*/
void CMiDirectorioMod::AddCategory(const TMiDirectorioCat&
aCategory)
 {
 iMiDirectorioCategories.Append(aCategory);
 iDirty = ETrue;
 }

/**Renombra una categoria*/
void CMiDirectorioMod::RenameCategory(const TInt aHandle, const
 TDesC& aNewName)
 {
 const TInt count = iMiDirectorioCategories.Count();
 for(TInt index = 0; index < count; index++)
       {
       if(iMiDirectorioCategories[index].CategoryHandle() ==
       aHandle)
             {

 iMiDirectorioCategories[index].SetCategoryName(aNewName);
             iDirty = ETrue;
             break;
             }
```

```
        }
 }

/**Elimina una categoría del array de categorias*/
void CMiDirectorioMod::RemoveCategory(const TInt aHandle)
 {
 const TInt count = iMiDirectorioCategories.Count();
 for(TInt index = 0; index < count; index++)
        {
        if(iMiDirectorioCategories[index].CategoryHandle() ==
        aHandle)
                {
                iMiDirectorioCategories.Remove(index);
                iDirty = ETrue;
                break;
                }
        }
 CategoryRemoved(aHandle);
 }

/**Devuelve el número de categorías que existen en el array*/
TInt CMiDirectorioMod::CategoryCount() const
 {
 return iMiDirectorioCategories.Count();
 }

/**Devuelve el id de una categoría específica*/
TMiDirectorioCat& CMiDirectorioMod::Category(const TInt aIndex)
 {
 return iMiDirectorioCategories[aIndex];
 }

void CMiDirectorioMod::CategoryRemoved(TInt aHandle)
 {
 const TInt itemCount = ItemCount();
 for(TInt index = 0; index < itemCount; index++)
        {
        CMiDirectorioIt* item = Item(index);
        if(item->Category() == aHandle)
                {
                item->SetCategory(EMiDirectorioCatUnfiled);
                }
        }
 // Comprueba si el ítem a eliminar existe y si su categoría es la
 //misma que la categoría a eliminar.
 if(iDeletedItem && iDeletedItem->Category() == aHandle)
        {
        iDeletedItem->SetCategory(EMiDirectorioCatUnfiled);
        }
 }

/**Comprueba si una categoría existe en el modelo. Develve Etrue si
existe y Efalse en caso contrario.*/
TBool CMiDirectorioMod::CategoryExists(TInt aHandle)
 {
        if((aHandle >= EMiDirectorioCatUnfiled) && (aHandle <=
        EMiDirectorioCatMusic))
        {
        return ETrue;
        }
```

```
// Comprube si existe la categoria
const TInt count = iMiDirectorioCategories.Count();
for(TInt index = 0; index < count; index++)
	{
	if(iMiDirectorioCategories[index].CategoryHandle() ==
	aHandle)
		{
		return ETrue;
		}
	}
return EFalse;
}

TUid CMiDirectorioMod::NewUniqueUid()
{
// Devuelve el ID mas
TUid uid;
uid.iUid = 0;

// Busca el ítem con ese id
const TInt count = iMiDirectorioIts.Count();
for(TInt index = 0; index < count; index++)
	{
	if(iMiDirectorioIts[index]->Uid().iUid > uid.iUid)
		{
		uid.iUid = iMiDirectorioIts[index]->Uid().iUid;
		}
	}
if(iDeletedItem && iDeletedItem->Uid().iUid > uid.iUid)
	{
	uid.iUid = iDeletedItem->Uid().iUid;
	}
uid.iUid++;
return uid;
}
```

Las funciones más relevantes son:

- *void CMiDirectorioMod::CategoryRemoved(TInt aHandle)*, esta función es llamada cuando una categoría quiere ser eliminada. Esta marca todos los ítems de dicha categoría como “Sin Categoría” y devuelve el control a la función que la llamo para que la elimine totalmente.
-
- *TUid CMiDirectorioMod::NewUniqueUid()*, Esta función crea un ID único para el próximo ítem que se genere dentro de Mi directorio. De esta manera nunca podrán coincidir dos ítems con el mismo ID.

4.4.5.4 Emulación de la aplicación con Carbide C++

A continuación se van a poner unas capturas de pantalla de la aplicación que se acaba de comentar funcionando sobre el emulador de Carbide C++ para poder ver la apariencia de la misma.

Figura 199. Captura de pantalla inicial de la aplicación. Como se puede ver al inicio de la misma no contiene ningún ítem.

Figura 200. Menú principal de la aplicación.

Figura 201. Menú de edición de archivo. Editando nombre.

Figura 202. Menú de edición de archivo. Editando año.

Figura 203. Menú de edición de archivo. Editando ranking

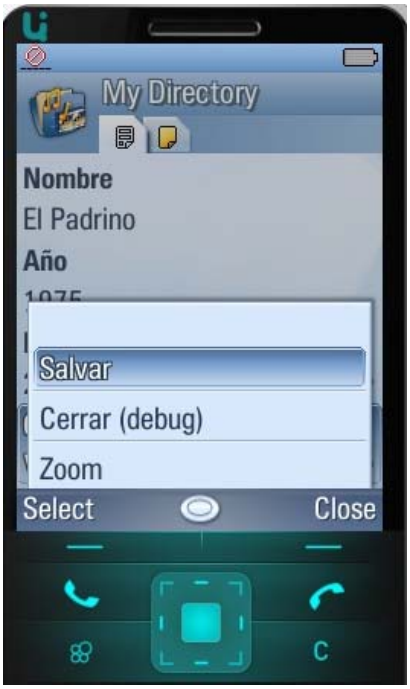


Figura 204. Menú de Archivo. Salvando Archivo

Figura 205. Pantalla con la ficha del Archivo

Figura 206. Menú desplegable de selección de Tipo de Archivo

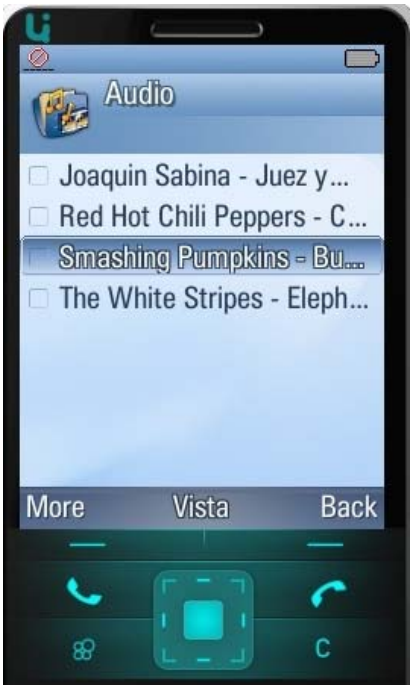


Figura 207. Pantalla con archivos del tipo Audio.

Figura 208. Pantalla con archivos de tipo Video.

Figura 209.Pantalla de confirmación de eliminación de archivo.

Figura 210. Pantalla con archivos de tipo Libros.

Figura 211. Pantalla con archivos de tipo Juegos

Figura 212. Menú para editar las categorías o para añadir nuevas categorías.

Figura 213. Pantalla principal de la aplicación donde se muestran a la vez todos los tipos de archivos ordenados alfabéticamente. Como se puede ver se pueden marcar o bien para enviar a otros dispositivos, para eliminar varios de una sola vez, para moverlos a otra carpeta o disco, etc.

Capitulo 5

Conclusión y líneas futuras

5.1 Conclusión

Este proyecto ha consistido en el desarrollo del conjunto de herramientas que otorga Symbian OS para realizar aplicaciones. Este sistema como se ha visto proporciona multitud de ellas y se han intentado explicar todas con mayor o menor profunidad.

El fin de este proyecto era dotar al lector de los conocimientos básicos para poder desarrollar aplicaciones con cualquiera de estas herramientas utilizando tanto sus diversos lenguajes, como los diferentes IDEs y todo lo que acompaña a los mismos. En muchos de los puntos quizá la profunidad con la que se han visto algunas de dichas herramientas es elevada pero esto era debido a la necesidad intrínseca de algunas de las aplicaciones desarrolladas.

Además de todas las herramientas se ha aprovechado para dar un repaso a las diversas tecnologías también necesarias a la hora de implementar las aplicaciones. Ya que un buen conocimiento de la tecnología y de su funcionamiento otorga al programador un campo de visión y un abanico de posibilidades mucho más elevado, con el fin de poder llevar a buen puerto los retos ofrecidos.

Con todo ello se espera facilitar el trabajo a todo aquel/aquella que quiera aprender a realizar aplicaciones en esta plataforma. Encontrandose en un punto de partida mucho más accesible. Se menciona esto porque la realización de este proyecto ha sido complicado debido a la inexactitud del material de trabajo existente hasta el momento, a la dispersión de la información acerca de Symbian OS y a la falta de manuales que permitiera un abordaje más lógico y factible.

5.2 Lineas futuras.

Este proyecto deja mucho margen a la hora de expandirlo, ya que a pesar de intentar abarcar lo máximo posible en su desarrollo siempre queda algo en el tintero que se podría haber visto mejor. Además por la fuente de este proyecto, Symbian OS, siendo esta una plataforma tan moderna y que esta tan a la orden del día, casi a diario se pueden encontrar nuevos recursos en los que incidir en futuros desarrollos.

Algunas posibles investigaciones futuras son:

- Manejo de redes inalámbricas (sobre todo Wi-Fi) en profundidad y desarrollo de aplicaciones utilizando dicha tecnología.
- Estudio con mayor profundidad de la plataforma de seguridad que ofrece Symbian OS.

Bibliografía

[1] ALLIN J. (2001): Wireless Java for Symbian Devices. Willey, England.

[2] HARRISON R. (2003): Symbian OS C++ for Mobile Phones. Wiley, England.

[3] Sony Ericsson Developer World. http://www.SonyEricsson.com/developer

[4] Nokia Developer. http://www.forum.nokia.com

[5] Symbian C++ Developer. http://www.symbian.com/developer

[6] Bluetooth Web Site. http://www.bluetooth.org

[7] NewLC. http://www.newlc.com

[8] Netbeans Web Site. http://www.netbeans.org

[9] UIQ Developer Comunity. https://developer.uiq.com/kb.html

[10] All About Symbian. http://allaboutsymbian.com

[11] Infrared Data Association. http://www.irda.org

[12] Wi-Fi Alliance. http://www.wi-fi.org

[13] Nokia NSeries. http://www.nseries.com/index.htm

[14] The Tech FAQ. http://www.tech-faq.com

[15] Barrapunto.com. http://barrapunto.com